51CTO.com
技术成就梦想
我 们 只 谈 开 发

开发月刊

Development Monthly

2013年10月
总第031期

Java下一代: 没有继承性的扩展

谁说设计师不会写代码？
—Photoshop脚本语言简介

DevOps进化论：新时代的土豪

从软件工程初期发展的独立主义到90年代开始强调了软件工程化到今天完全进化成为一个产业链，这就是我们现在谈的DevOps。它是如何盛行、本质是什么、共性在哪里、怎么认识它的落地？这就是今天小编和大家一起讨论的话题。

51CTO.com
技术成就梦想

开发月刊

出版方：

北京无忧创想信息技术有限公司

责任编辑：

林师授

封面设计：

钱靓

联系方法：

邮箱：linss@51cto.com

电话：010-68476606-8123

出版日期：2013年10月23日

欢迎来稿，请发送邮件至

linss@51cto.com

主编的话

本期《开发月刊》的专题内容是Java下一代语言Groovy、Scala 和 Clojure，它们的共同点多于不同点，主要集中于很多功能和方便性上的共同点。专题文章探究它们各自如何克服 Java语言中长期存在的一个缺点—无法重载操作符。还要讨论关联性和优先级等相关概念。月刊以原创译文为主，把国外最新，最热的技术展现给大家。

同时，开发月刊集合了热门的资源下载，最近热门技术专题以及技术热点资讯。不管朋友们喜不喜欢我们推荐的文章，但是我们会一如既往的把开发最好的内容推荐给网友，与网友共同进步，成长。

祝好！

51CTO开发频道寄语

目录 Contents

2013年10月
总第31期

编程排行 Billboard

专题报道 Special report

原创译文 Original translation

技术热点 Techlogy hot

编者按

TIOBE社区今天发布的2013年10月的编程语言排行榜，轻量级Java语言Groovy在本期榜单中排在第18位，取得了历史性突破，首次闯入排行榜前二十。前五名内没有太大的变化，C语言岿然不动，Java紧随其后。

2013年10月编程语言排行榜：Groovy首次闯入前二十

TIOBE社区今天发布的2013年10月的编程语言排行榜，轻量级Java语言Groovy在本期榜单中排在第18位，取得了历史性突破，首次闯入排行榜前二十。前五名内没有太大的变化，C语言岿然不动，Java紧随其后。和上月相比，只是排在第三位的C++和排在第四位的Objective-C换了位置，小编上周的猜测貌似是应验了。上周的黑马Transact-SQL下降一位，不过势头依然迅猛，和去年的同月相比提高了14位。

Groovy在本月击败了其他知名的JVM语言，比如Scala（36位），JavaFX脚本（第41）和Clojure（#76）。让我们拭目以待，看它是否能在未来几个月保持其前20名的位置。

前20名榜单排行榜

Position Oct 2013	Position Oct 2012	Delta in Position	Programming Language	Ratings Oct 2013	Delta Oct 2012	Status
1	1	=	C	17.246%	-2.58%	A
2	2	=	Java	16.107%	-1.09%	A
3	3	=	Objective-C	8.992%	-0.49%	A
4	4	=	C++	8.664%	-0.60%	A
5	6	↑	PHP	6.094%	+0.43%	A
6	5	↓	C#	5.718%	-0.81%	A
7	7	=	(Visual) Basic	4.819%	-0.30%	A
8	8	=	Python	3.107%	-0.79%	A
9	23	↑↑↑↑↑↑↑↑↑↑↑↑↑↑	Transact-SQL	2.621%	+2.13%	A
10	11	↑	JavaScript	2.038%	+0.78%	A
11	18	↑↑↑↑↑↑↑	Visual Basic .NET	1.933%	+1.33%	A
12	9	↓↓↓	Perl	1.607%	-0.52%	A
13	10	↓↓↓	Ruby	1.246%	-0.56%	A
14	14	=	Pascal	0.753%	-0.09%	A
15	17	↑↑	PL/SQL	0.730%	+0.10%	A
16	13	↓↓↓	Lisp	0.725%	-0.22%	A
17	12	↓↓↓↓↓	Delphi/Object Pascal	0.701%	-0.40%	A
18	53	↑↑↑↑↑↑↑↑↑↑↑↑↑↑

21-50编程语言排名：

Position	Programming Language	Ratings
21	R	0.553%
22	SAS	0.543%
23	Ada	0.510%
24	F#	0.499%
25	Fortran	0.474%
26	Assembly	0.471%
27	Bash	0.470%
28	Ladder Logic	0.457%
29	Logo	0.433%
30	Lua	0.413%
31	ABAP	0.394%
32	C shell	0.382%
33	Common Lisp	0.380%
34	NXT-G	0.366%
35	Scheme	0.360%
36	Scala	0.345%
37	D	0.337%
38	Prolog	0.328%
39	RPG (OS/400)	0.319%
40	PostScript	0.312%
41	JavaFX Script	0.297%
42	Tcl	0.294%
43	Erlang	0.292%
44	Max/MSP	0.276%
45	Scratch	0.270%
46	Haskell	0.245%
47	ML	0.245%
48	PL/I	0.240%
49	ActionScript	0.215%
50	Emacs Lisp	0.210%

关于Groovy

Groovy 是用于Java虚拟机的一种敏捷的动态语言，它是一种成熟的面向对象编程语言，既可以用于面向对象编程，又可以用作纯粹的脚本语言。使用该种语言不必编写过多的代码，同时又具有闭包和动态语言中的其他特性。

Groovy是JVM的一个替代语言（替代是指可以用 Groovy 在Java平台上进行Java 编程），使用方式基本与使用 Java代码的方式相同，该语言特别适合与Spring的动态语言支持一起使用，设计时充分考虑了Java集成，这使 Groovy 与 Java 代码的互操作很容易。（注意：不是指Groovy替代java，而是指Groovy和java很好的结合编程。）

Groovy是一个被编译成字节码的面向对象的语言。它的语法风格与java很像，但是又没java那么混乱。Java程序员能够很快的熟练使 用 Groovy；实际上，Groovy编译器是可以接受完全纯粹的Java语法格式的，这样能够让程序员在学习Groovy时不需要学习任何新语法。需要注 意使用Groovy的一个重要特点就是使用类型推断，即能够让编译器能够在程序员没有明确说明的时候推断出变量的类型。同时Groovy也提供了一个高级 架构来解决字符串处理，xml的序列化与反序列化以及单元测试等等 -- 所有的这一切好处都能为程序员节约不少时间。

9个杀手级 JVM 编程语言

Java虚拟机已经不再是仅仅局限在 Java 了，很多语言提供了脚本转换，可以让其他的程序在java虚拟机上运行，这样能够让更多的开发者能够依靠JVM在Java平台上大有作为。而且虚拟机以及对应的字节码都是开源的，所以你也能很容易地生成对应字节码来做一款属于你自己的编程语言。我们今天来看看以下9种新的编程语言。

编者按

Groovy、Scala和Clojure提供了许多扩展机制，但继承几乎是Java语言的惟一选择。这一期将介绍类别类、ExpandoMetaClass、隐式转换和协议，借助它们来使用 Java 下一代语言扩展 Java 类。

Java下一代: 没有继承性的扩展

Java 语言的设计有目的地进行了一定的删减，以避免前代产品中已发现的一些问题。例如，Java 语言的设计人员感觉 C++ 中的多重继承性带来了太多复杂性，所以它们选择不包含该特性。事实上，他们在该语言中很少构建扩展性选项，仅依靠单一继承和接口。

其他语言（包括 Java 下一代语言）存在巨大的扩展潜力。在本期和接下来的两期文章中，我将探索扩展 Java 类而不涉及继承性的途径。在本文中，您会了解如何向现有类添加方法，无论是直接还是通过语法糖 (syntactic sugar)。

表达式问题

表达式问题是最近的计算机科学历史上的一个众所周知的观察结果，首创于贝尔实验室的 Philip Wadler 的一篇未发表的论文（参见 参考资料）。（Stuart Sierra 在其 developerWorks 文章 “通过 Clojure 1.2 解决表达式问题” 中出色地解释了它。在这篇文章中，Wadler 说道:

表达式问题是老问题的新名字。我们的目标是通过案例定义数据类型，在这里，在不重新编译现有代码的情况下，您可以将新的案例添加到数据类型和数据类型的新函数中，同时保留静态类型安全（例如，没有转换）。

换句话说，您如何向一个分层结构中的类添加功能，而不求助于类型转换或 if 语句?

我们将通过一个简单的例子来表明表达式问题在真实世界中的表现形式。假设您公司始终假设应用程序中的长度单位为米，没有在您的类中为任何其他长度单位构建任何功能。但是，有一天，您公司与一家竞争对手合并了，而这个竞争对手始终假设长度单位为英尺。

解决该问题的一种方法是，通过使用转换方法扩展 Integer，使两种格式之间的切换变得无关紧要。现代语言提供了多种解决方案来实现此目的；在本期中，我将重点介绍其中的 3 种:

- 开放类
- 包装器类
- 协议

Groovy 的类别和 ExpandoMetaClass

Groovy 包含两种使用开放类 扩展现有的类的不同方式，“重新开放” 一个类定义来实现更改（例如添加、更改或删除方法）的能力。

类别类

类别类（一种借鉴自 Objective-C 的概念）是包含静态方法的常规类。每个方法至少接受一个参数，该参数表示方法扩充的类型。如果希望向

Integer 添加方法，例如我需要接受该类型作为第一个参数的静态方法，如清单 1 所示：

清单 1. Groovy 的类别类

```
1. class IntegerConv {
2.   static Double getAsMeters(Integer self) {
3.     self * 0.30480
4.   }
5.    static Double getAsFeet(Integer self) {
6.     self * 3.2808
7.   }
8. }
```

清单 1 中的 IntegerConv 类包含两个扩充方法，每个扩充方法都接受一个名为 self（一个通用的惯用名称）的 Integer 参数。要使用这些方法，我必须将引用代码包装在一个 use 代码块中，如清单 2 所示：

清单 2. 使用类别类

```
1. @Test void test_conversion_with_category() {
2.   use(IntegerConv) {
3.     assertEquals(1 * 3.2808, 1.asFeet, 0.1)
4.     assertEquals(1 * 0.30480, 1.asMeters, 0.1)
5.   }
6. }
```

清单 2 中有两个特别有趣的地方。首先，尽管清单 1 中的扩展方法名为 getAsMeters()，但我将它称为 1.asMeters。Groovy 围绕 Java 中的属性的语法糖使我能够执行 getAsMeters() 方法，好像它是名为 asMeters 的类的一个字段一样。如果我在扩展方法中省略了 as，对扩展方法的调用需要使用空括号，就像 1.asMeters() 中一样。一般而言，我喜欢更干净的属性语法，这是编写特定于域的语言 (DSL) 的一种常见技巧。

清单 2 中第二个需要注意的地方是对 asFeet 和 asMeters 的调用。在 use 代码块中，我同等地调用新方法和内置方法。该扩展在 use 代码块的词法范围内是透明的，这很好，因为它限制了扩充（有时是一些核心）类的范围。

ExpandoMetaClass

类别是 Groovy 添加的第一种扩展机制。但事实证明对构建 Grails（基于 Groovy 的 Web 框架）而言，Groovy 的词法范围限制太多了。由于不满类别中的限制，Grails 的创建者之一 Graeme Rocher 向 Groovy 添加了另一种扩展机制：ExpandoMetaClass。

ExpandoMetaClass 是一种懒惰实例化的扩展持有者，它可从任何类 “成长” 而来。清单 3 展示了如何使用 ExpandoMetaClass，为我的 Integer 类实现我的扩展：

清单 3. 使用 ExpandoMetaClass 扩展 Integer

```
1. class IntegerConvTest{
2.     static {
3.       Integer.metaClass.getAsM { ->
4.         delegate * 0.30480
5.       }
6.       Integer.metaClass.getAsFt { ->
7.         delegate * 3.2808
8.       }
9.    }
10.      @Test void conversion_with_expando() {
11.          assertTrue 1.asM == 0.30480
```

```
12.     assertTrue 1.asFt == 3.2808
13.       }
14.     }
```

在 清单 3 中，我使用 metaClass holder 添加 asM 和 asFt 属性，采用与 清单 2 相同的命名约定。对 metaclass 的调用出现在测试类的一个静态初始化器中，因为我必须确保扩充操作在遇到扩展方法之前发生。

类别类和 ExpandoMetaClass 都在内置方法之前调用扩展类方法。这使您能够添加、更改或删除现有方法。清单 4 给出了一个示例：

清单 4. 取代现有方法的扩展类

```
1. @Test void expando_order() {
2.   try {
3.     1.decode()
4.   } catch(NullPointerException ex) {
5.     println(“can’t decode with no
   parameters”)
6.   }
7.   Integer.metaClass.decode { ->
8.     delegate * Math.PI;
9.   }
10.       assertEquals(1.decode(), Math.PI, 0.1)
11.     }
```

清单4 中的第一个 decode() 方法调用是一个内置的静态 Groovy 方法，它设计用于更改整数编码。正常情况下，它会接受一个参数；如果调用时没有任何参数，它将抛出 NullPointerException。但是，当我使用自己的 decode() 方法扩充 Integer 类时，它会取代原始类。

Scala 的隐式转换

Scala 使用包装器类 来解决表达式问题的这个方面。要向一个类添加一个方法，可将它添加到一个帮助类中，然后提供从原始类到您的帮助器的隐式转换。在执行转换之后，您就可以从帮助器隐式地调用该方法，而不是从原始类调用它。清单 5 中的示例使用了这种技术：

清单 5. Scala 的隐式转换

```
1. class UnitWrapper(i: Int) {
2.   def asFt = {
3.     i * 3.2808
4.   }
5.   def asM = {
6.     i * 0.30480
7.   }
8. }
9.  implicit def unitWrapper(i:Int) = new
   UnitWrapper(i)
10.       println(“1 foot = “ + 1.asM + “
   meters”);
11.       println(“1 meter = “ + 1.asFt + “foot”)
```

在清单5中，我定义了一个名为 UnitWrapper 的帮助器类，它接受一个构造函数参数和两个方法：asFt 和 asM。在拥有转换值的帮助类后，我创建了一个 implicit def，实例化一个新的 UnitWrapper。要调用该方法，可以像调用原始类的一个方法那样调用它，比如 1.asM。当 Scala 未在 Integer 类上找到 asM 方法时，它会检查是否存在隐式转换，从而允许将调用类转换为一个包含目标方法的类。像 Groovy 一样，Scala 拥有语法糖，因此我能够省略方法调用的括号，但这是一种语言特性而不是命名约定。

Scala 中的转换帮助器通常是 object 而不是类，但我使用了一个类，因为我希望传递一个值作为构造函数参数（object 不允许这么做）。

Scala 中的隐式转换是一种扩充现有类的精妙且类型安全的方式，但不能向开放类一样，使用这种机制更改或删除现有方法。

Clojure 的协议

Clojure 采用了另一种方法来解决表达式问题的这个方面，那就是结合使用 extend 函数和 Clojure 协议 抽象。协议在概念上类似于一个 Java 接口：一个没有实现的方法签名集合。尽管 Clojure 实质上不是面向对象的，而是偏向于函数，但您可以与类进行交互（并扩展它们），并将方法映射到函数。

为了扩展数字以添加转换，我定义了一个协议，它包含我的两个函数（asF 和 asM）。我可使用该协议 extend 一个现有类（比如 Number）。extend 函数接受目标类作为第一个参数，接受该协议作为第二个参数，以及一个使用函数名为键并使用实现（以匿名函数形式）为值的映射。清单 6 显示了 Clojure 单位转换：

清单 6. Clojure 的扩展协议

```
1. (defprotocol UnitConversions
2.   (asF [this])
3.   (asM [this]))
4. (extend Number
5.   UnitConversions
6.   {:asF (fn [this] (* this 3.2808))
7.    :asM #(* % 0.30480)})
```

我可以在 Clojure REPL（interactive read-eval-print loop，交互式读取-重新运算-打印循环）上使用新的扩展来验证该转换：

```
1. user=> (println "1 foot is " (asM 1) "
   meters")
2. 1 foot is 0.3048 meters
```

在 清单 6 中，两个转换函数的实现演示了匿名函数声明的两种语法变体。每个函数只接受一个参数（asF 函数中的 this）。单参数函数很常见，以至于 Clojure 为它们的创建提供了语法糖，如 AsM 函数中所示，其中 % 是参数占位符。

协议创建了一种将方法（以函数形式）添加到现有类中的简单解决方案。Clojure 还包含一些有用的宏，使您能够将一组扩展整合在一起。例如，Compojure Web 框架（参见 参考资料）使用协议扩展各种类型，以便它们 "知道" 如何呈现自身。清单 7 显示了来自 Compojure 中的 Renderable 的一段代码：

清单 7. 通过协议扩展许多类型

```
1. (defprotocol Renderable
2.   (render [this request]
3.     "Render the object into a form suitable for
   the given request map."))
4. (extend-protocol Renderable
5.   nil
6.   (render [_ _] nil)
7.   String
8.   (render [body _]
9.     (-> (response body)
10.          (content-type "text/html;
   charset=utf-8")))
11.       APersistentMap
12.       (render [resp-map _]
```

```
13.        (merge (with-meta (response “”)
   (meta resp-map))
14.                resp-map))
15.       IFn
16.       (render [func request]
17.         (render (func request)
18.       ; . . .
```

在 清单 7 中，Renderable 协议是使用单个 render 函数来定义的，该函数接受一个值和一个请求映射作为参数。Clojure 的 extend-protocol 宏（它可用于将协议定义分组到一起）接受类型和实现对。在 Clojure 中，您可使用下划线代替不关心的参数。在 清单 7 中，这个定义的可看见部分为 nil、String、APersistentMap 和 IFn（Clojure 中的函数的核心接口）提供了呈现指令。（该框架中还包含其他许多类型，但为节省空间，清单中省略了它们。）可以看到这在实践中非常有用：对于您可能需要呈现的所有类型，您可将语义和扩展放在一起定义。

结束语

在本期中，我介绍了表达式问题，剖析了 Java 下一代语言如何处理以下方面：现有类的干净扩展。每种语言都使用一种不同的技术（Groovy 使用开放类，Scala 使用包装器类，而 Clojure 实现了协议）来实现类似的结果。

但是，表达式问题比类型扩充更深刻。在下一期中，我将继续讨论使用其他协议功能、特征和 mix-in 的扩展。 ■

本文链接：
http://developer.51cto.com/art/201310/412670.htm

参考资料

学习

- Scala：Scala是JVM上一种现代的函数式语言
- Clojure：Clojure是在 JVM 上运行的一种现代的函数式语言。
- Groovy是Java的一个动态变体，拥有更新的语法和功能。
- “表达式问题”：Philip Walder 1998 年为发表的论文详细介绍了表达式问题。
- “通过 Clojure 1.2 解决表达式问题”（Stuart Sierra，developerWorks，2010 年 12 月）：了解 Clojure 的表达式问题解决方案的更多信息。
- Compojure：Compojure是一个使用Clojure编写的 Ring 路由框架。
- 探索Java平台的替代语言：跟随此知识路径，研究关于各种替代性 JVM 语言的 developerWorks 内容。
- 语言设计者的笔记本：在这个developerWorks系列中，Java 语言架构师 Brian Goetz 探讨了一些语言设计问题，这些问题为 Java 语言在 Java SE 7、Java SE 8 和更改版本中的演化带来了挑战。
- 函数式思维：在 developerWorks 上 Neal Ford 的专栏系列中探索函数式编程。
- 该作者的更多文章（Neal Ford，developerWorks，2005 年 6 月至今）：了解 Groovy、Scala、Clojure、函数式编程、架构、设计、Ruby、Eclipse 和其他 Java 相关技术。

Java下一代:Groovy、Scala和Clojure共同点 第 1 部

探究这些下一代 JVM 语言如何处理操作符重载

■ Java 下一代语言（Groovy、Scala 和 Clojure）的共同点多于不同点，主要集中于很多功能和方便性上的共同点。本期文章探究它们各自如何克服 Java™ 语言中长期存在的一个缺点 — 无法重载操作符。还要讨论关联性和优先级等相关概念。

编程语言中的好理念可以延续并扩展到其他语言，就像美酒一样历久弥香。因此，不足奇怪的是，Java 下一代语言 — Groovy、Scala 和 Clojure — 具有很多共同的特性。在本期和下一期 Java 下一代 文章中，我将探讨每种语言语法中功能清单的一致性。我从能够重载操作符这个特性说起 — 克服了Java 语言中长期存在的一个缺点。

操作符重载

如果您改造过 Java BigDecimal 类，可能看到过类似于清单 1 的代码：

清单1. Java 代码中的 Lackluster BigDecimal 支持

```
1. BigDecimal op1 = new BigDecimal(1e12);
2. BigDecimal op2 = new BigDecimal(2.2e9);
3. // (op1 + (op2 * 2)) / (op1/(op1 + (op2 *
   1.5e2))
4. BigDecimal lhs = op1.add(op2.
   multiply(BigDecimal.valueOf(2)));
5. BigDecimal rhs = op1.divide(
6.         op1.add(op2.multiply(BigDecimal.
   valueOf(1.5e2))),
7.             RoundingMode.HALF_UP);
8. BigDecimal result = lhs.divide(rhs);
9. System.out.println(String.format( “%,.2f”,
   result));
```

在 清单 1 中，我试图实现注释中的这个公式。在 Java 编程中，因无法重载数学操作符，使得我只能求助于方法调用。静态导入可以解决问题，但是对于所选择的上下文，显然需要适当的操作符重载。最初的 Java 工程师故意从语言上忽略操作符重载，不过这感觉增加了太大的复杂性。但是经验表明，因缺乏这一特性而强加给开发人员的复杂性更甚于潜在的滥用机会。

用稍微各不相同的方式，所有三种 Java 下一代语言都实现了操作符重载。

Scala 的操作符

Scala 通过放弃操作符与方法之间的区别而允许操作符重载。操作符只不过是具有特殊名称的方法。例如，要重写乘法操作符，可以重写 * 方法。[* 是一个有效的方法名称，这就是 Scala 使用下划线 (_) 符号而不是 Java 星号 (*) 符号来代表导入的原因之一。]

我使用复数来说明重载。复数是一种数学表示，包括实部和虚部，例如通常写作 3 + 4i 这样的形式（参见 参考资料）。复数在很多科学领域都很

常见，包括工程学、物理学、电磁学以及其他理论。清单 2 显示了复数的 Scala 实现：

清单 2. Scala 复数

```
1. final class Complex(val real:Int, val imaginary:Int)
   {
2.   require (real != 0 || imaginary != 0)
3.   def +(operand:Complex) =
4.       new Complex(real + operand.real, imaginary
   + operand.imaginary)
5.   def +(operand:Int) =
6.    new Complex(real + operand, imaginary)
7.   def -(operand:Complex) =
8.    new Complex(real - operand.real, imaginary
   - operand.imaginary)
9.   def -(operand:Int) =
10.          new Complex(real - operand, imaginary)
11.        def *(operand:Complex) =
12.           new Complex(real * operand.real -
   imaginary * operand.imaginary,
13.                real * operand.imaginary +
   imaginary * operand.real)
14.        override def toString() =
15.           real + (if (imaginary < 0)  “”  else
   “+” ) + imaginary +  “i”
16.        override def equals(that:Any) = that
   match {
17.          case other :Complex => (real == other.
   real) && (imaginary == other.imaginary)
18.          case _ => false
19.        }
20.        override def hashCode():Int =
21.          41 * ((41 + real) + imaginary)
22.      }
```

Scala 通过折叠不必要的脚手架代码，大大降低了 Java 语言的啰嗦程度。例如，在 清单 2 中，类中的构造函数参数和字段与类定义一起出现。在本例中，类的主体充当构造函数，所以对 require() 方法的调用在第一次实例化操作过程中验证值的存在。因为 Scala 自动提供字段，所以类的其余部分包含方法定义。对于 +、- 和 * 操作符，我都声明了接受 Complex 数作为参数的同名方法。复数的乘法不及加法和减法那么直观。清单 2 中已重载的 * 方法实现公式：

1. (x + yi)(u + vi) = (xu - yv) + (xv + yu)i

equals() 和 match 关键字

清单 2 中另一个有趣的特性是在 equals() 方法中使用了模式匹配。尽管 Scala 中支持强制类型转换，但是类型匹配更为常见。that 参数被声明为 Any — Scala 继承层次的顶层。该方法的主体由 match 调用组成，在传递的类型匹配时，该调用检查实部和虚部的值，否则默认为 false。

清单 2 中的 toString() 方法例示了 Java 下一代语言之间的另外一个共同点：使用表达式而不是语句。在 toString() 方法中，虚部为正时我必须提供加号 (+)，否则，虚部的隐式减号就足够了。在 Scala 中，if 是一个表达式，而不是语句，不再需要 Java 三元操作符 (?:)。

实际上，增加的 +、- 和 * 方法都跟标准的操作符没什么区别，如清单 3 中的单元测试所示：

清单 3. 练习 Scala 复数

```
1. class ComplexTest extends FunSuite {
```

```
2.  test(“addition”) {
3.    val c1 = new Complex(1, 3)
4.    val c2 = new Complex(4, 5)
5.    assert(c1 + c2 === new Complex(1+4, 3+5))
6.  }
7.  test(“subtraction”) {
8.    val c1 = new Complex(1, 3)
9.    val c2 = new Complex(4, 5)
10.         assert(c1 - c2 === new Complex(1-4,
    3-5))
11.       }
12.       test(“multiplication”) {
13.         val c1 = new Complex(1, 3)
14.         val c2 = new Complex(4, 5)
15.         assert(c1 * c2 === new Complex(
16.             c1.real * c2.real - c1.imaginary * c2.
    imaginary,
17.             c1.real * c2.imaginary + c1.imaginary
    * c2.real))
18.       }
19.     }
```

清单 3 中的测试失败，揭示了一个有趣的不一致性。后面讨论 关联性 时，我指出并解决了这个问题。但是，现在简单介绍一下 Groovy 和 Clojure 中的重载。

Groovy 的映射

通过提供您可以重写的映射方法，Groovy 重载任何 Java 操作符。（例如，要重写 + 操作符，您可在 Integer 类重写 plus() 方法。）在 “函数设计模式，第 3 部分”，即我 函数式思维 系列（探讨函数语言中的可扩展性）中的一期文章，我用同一个复数例子详细介绍了 Groovy 的操作符重载。

在 Groovy 中，您无法创建新的操作符（尽管可以创建新方法）。一些框架（比如 Spock 测试框架；参见 参考资料）重载难以理解却实际存在的操作符，比如 >>>。Scala 和 Clojure 都更加一致地对待操作符和方法，尽管方式有所不同。

Groovy 也引入了几个方便的新操作符，比如 ?. 和 Elvis 操作符 (?:)，—前者是安全导航 操作符，它确保所有调用者都不为空，后者是 Java 三元操作符的简写形式，对于轻松提供默认值非常有用。Groovy 对新操作符没有扩展方法，防止了开发人员重载它们。至于开发人员为什么想要重载它们，原因不是很清楚：操作符重载的一个基本原因在于，以前的操作符使用经验可以增加代码的可读性。您不可能在 Groovy 外面培养这些操作符的使用经验。如果您为方便性使用操作符时破坏了代码可读性，那么操作符重载将变成危险的事情。

Clojure 的操作符

跟 Scala 中一样，Clojure 中的操作符也只是带有符号名称的方法。因此，比如说您可以随便为自己的定制类型创建一个 + 方法。然而，要在 Clojure 中正确重写操作符，您必须理解协议 和一种用于从公共内核生成一组方法的技术。我将在下一期文章中讨论这一内容。

关联性

操作符关联性 是指操作符是等式左侧还是右侧的方法。Scala 对空格的使用不同于大多数其他语言，因为基本上任何 Scala 方法都可以充当操作符。例如，表达式 x + y 实际上就是方法调用 x.+(y)，如清单 4 中 Scala REPL（解释器）会话

中所示：

清单 4. Scala 中的空格转化

```
1. scala> val sum1 = x.+(y)
2. sum1:Int = 22
3.
4. scala> val sum2 = (12).+(10)
5. sum2:Int = 22
```

清单 4 中可以看到，空格转化也适用于常量。愿意的话，您可以将 Scala 中的所有方法都看作操作符。例如，String 类具有一个 indexOf() 方法，它返回被作为参数传递的字符串中的索引位置。在 Scala 中，您可以用传统方式通过 s.indexOf(‘a’) 调用过它，或者作为操作符 — 像 s indexOf ‘a’ 中一样。（这个具体的方法很有趣，因为它有一个已重载的版本，接受一个额外的参数来指定搜索开始处的索引位置。您仍然可以使用操作符表示法调用它，但是必须将参数放置在括号中，就像 s indexOf(‘a’, 3) 中一样。）

Groovy 遵循 Java 关联性约定，所以特定操作符的规则由语言定义。Clojure 根本不关注关联性；它的 Lisp 语法不依赖于关联性，因为所有语句都是意义明确的。

由于 Scala 的目标之一就是允许开发人员可以将任何东西都用作操作符，所以它不能依赖于任何关联性规则。该语言如何才能允许特殊的操作符却仍然建立规则？Scala 以一种支持开发人员最大自由度的创新方式解决了这个问题 — 使用操作符命名约定。默认情况下，Scala 中操作符是左关联的：表达式分解为一个对左操作数的方法调用，例如，这意味着表达式 x + y 分解为 x.+(y)。然而，如果方法名称以 : 结尾，则操作符是右关联的。例如，i +: j 调用转化成 j.+:(i)。

关联性解释了为什么 清单 3 中的测试无法得到正确的结果。清单 2 中的 Scala Complex 定义中，我实现了 + 和 - 操作符的版本，它们既接受 Complex，也接受 Int 参数类型。这种类型的灵活性允许复数与一般整数（即实部为零的复数）相互操作。清单 5 说明了单元测试中的互操作性：

清单 5. 混合类型的测试

```
1. test(“mixed addition from Complex”) {
2.   val c1 = new Complex(1, 3)
3.   assert(new Complex(7, 3) == c1 + 6)
4. }
5.
6. test(“mixed subtraction from
   Complex”) {
7.   val c1 = new Complex(10, 3)
8.   assert(new Complex(5, 3) == c1 - 5)
9. }
```

清单 5 中的两个测试都能通过，没有问题 — 操作符方法的 Int 版本开始了。然而，如果我尝试以下测试，它则会失败：

```
1. test(“mixed subtraction from Int”) {
2.   val c1 = new Complex(10, 3)
3.   assert(new Complex(15, 3) == 5 + c1)
4. }
```

两个测试之间的细微差别就在于关联性上。记住，在本例中，Scala 调用左操作符的方法，这意味着它试图开始一个为 Int 定义的方法（它知道如何处理复数）。

为了解决这个问题，我在 Int 和 Complex 之间定义了一个隐式强制类型转换。有多种方式展示这种转换，我将在以后几期文章中更加详细地介绍。在本例中，我创建了一个伴生对象，即 Complex，这是一个用于放置 Java 语言中声明为 static 的方法的地方：

```
1. final object Complex {
2.   implicit def intToComplex(x:Int) = new
   Complex(x, 0)
3. }
```

该定义包含单个方法，此方法接受一个 Int 并将之返回为 Complex。将这个声明作为 Complex 类放置在相同的源文件中，然后我通过 import nealford.javaNext.complexnumbers.Complex.intToComplex 命令在我的测试案例中导入该方法，可以支持隐式转换。有了转换之后，测试案例成功通过，因为测试知道如何处理通过操作符发出的方法调用。

优先级

操作符优先级（或者操作顺序）是指规定潜在存在歧义的情况下操作发生顺序的语言规则。对于公共操作符，Groovy 依赖于 Java 优先级规则；对于自己的定制操作符，它定义自己的规则。Clojure 不具有或不需要优先级规则；因为所有代码都以括号形式编写，不再会出现中缀表示法中固有的歧义性。

Scala 使用操作符名称的第一个字符来确定操作顺序，优先层次是：

- 所有其他特殊符号
- * / %
- + -
- :
- = !
- < >
- &
- ^
- |
- 所有字母
- 所有分配操作符

以较高级别字符开始的操作符具有较高的优先级。例如，表达式 x *** y ||| z 将分解为 (x.***(y)).|||(z)。该规则惟一的例外是分配语句，或者任何以等号 (=) 结尾的操作符，它们自动具有最低优先级。

结束语

Java 下一代语言的一个共同目标是，简化那些影响着 Java 语言的繁琐限制。操作符重载是每种语言解决这个问题的一个重要途径。所有三种语言都允许操作符重载，只是实现的方式有所不同。处理关联性和优先级这类问题的方式的细微差别表明了，各个语言部分是如何紧密联系的。Clojure 的有趣方面之一是它的语法 — 因为每个表达式都是括号形式的 — 消除了优先级和关联性中的歧义。

在下一期文章中，我将探究 “一切都是对象” 这一说法在 Java 下一代语言中的深层含义。■

文章链接：
http://developer.51cto.com/art/201305/396034.htm

Java下一代:Groovy、Scala和Clojure共同点 第 2 部分

了解Java 下一代语言如何减少样板代码和降低复杂性

■ 与 Java™ 语言相关的常见抱怨包括：简单的任务涉及到太多的步骤，默认设置有时难以理解。所有 3 种 Java 下一代语言在这些领域都采取了更加明智的方法。这一期 Java 下一代 展示了 Groovy、Scala 和 Clojure 如何消除 Java 语言的瑕疵。

Java 编程语言诞生时所面临的限制与如今的开发人员所面临的条件有所不同。具体来讲，由于上世纪 90 年代中期的硬件的性能和内存限制，Java 语言中存在原语类型。从那时起，Java 语言不断在演化，通过自动装箱（autobox）消除了许多麻烦操作，而下一代语言（Groovy、Scala 和 Clojure）更进一步，消除了每种语言中的不一致性和冲突。

在这一期的文章中，我将展示下一代语言如何消除一些常见的 Java 限制，无论是语法上还是默认行为上。第一个限制是原语数据类型的存在。

原语的消亡

Java 语言最开始有 8 对原语和相应的类型包装器类（最初用于解决性能和内存限制），并通过自动装箱逐步地淡化了它们之间的区别。Java 下一代语言更进一步，让开发人员觉得好像根本不存在差别。

Groovy 完全隐藏了原语类型。例如，int 始终表示 Integer，Groovy 自动处理数字类型的上变换，防止出现数值溢出错误。例如，请查看清单 1 中的 Groovy shell 交互：

清单 1. Groovy 对原语的自动处理

```
1. groovy:000> 1.class
2. ===> class java.lang.Integer
3. groovy:000> 1e12.class
4. ===> class java.math.BigDecimal
```

在 清单1中，Groovy shell 显示，即使是常量也是通过底层的类来表示的。因为所有数字（和其他伪装的原语）都是真正的类，所以可以使用元编程技术。这些技术包括将方法添加到数字中（这通常用于构建特定领域的语言，即 DSL），支持 3.cm 这样的表达式。在后面介绍可扩展性的那期文章中，我会更全面地介绍此功能。

与 Groovy 中一样，Clojure 自动屏蔽原语与包装器之间的区别，允许对所有类型执行方法调用，自动处理容量的类型转换。Clojure 封装了大量底层优化，这已在语言文档中详细说明（参阅 参考资料）。在许多情况下，可提供类型 hints，使编译器能够生成更快的代码。例如，无需使用 (defn sum[x] ...) 定义方法，可以添加一个类型提示，比如 (defn sum[^float x] ...)，它会为临界区 (critical section) 生成更高效的代码。

Scala 也屏蔽了原语之间的区别，通常对代码的时效性部件使用底层原语。它还允许在常量上调用

方法，就像 2.toString 中一样。借助其混搭原语和包装器的能力，比如 Integer，Scala 比 Java 自动装箱更加透明。例如，Scala 中的 == 运算符可在原语和对象引用上正确运行（比较值，而不是引用），而不同于相同运算符的 Java 版本。Scala 还包含一个 eq 方法（以及一个对称的 ne 方法），它始终比较底层引用类型是否等效。基本而言，Scala 会智能地切换默认行为。在 Java 语言中，== 会对引用数据进行比较，您几乎不需要这么做，可以使用不太直观的 equals() 比较值。在 Scala 中，== 能正确运行（比较值），无论底层实现是什么，它还提供了一个方法来执行不太常见的引用相等性检查 (reference equality check)。

Scala 的这一特性表明，Java 下一代语言的一个重要优势在于：将低级细节卸载到语言和运行时，开发人员能够有更多的时间考虑更高级的问题。

简化默认行为

人们的看法高度一致，大部分 Java 开发人员都认为，在 Java 语言中常见的操作需要太多的语法。例如，属性定义和其他样板代码使类定义变得很杂乱，掩盖了重要的方法。所有 Java 下一代语言都提供了简化创建和访问过程的途径。

Scala 中的类和 case 类

Scala 已简化了类定义，可为您自动创建存取函数、赋值函数和构造函数。例如，请查看清单 2 中的 Java 类：

清单 2. Java 中简单的 Person 类

```
1. class Person {
2.     private String name;
3.     private int age;
4.     Person(String name, int age) {
5.         this.name = name;
6.         this.age = age;
7.     }
8.     public String getName() {
9.         return name;
10.         }
11.         public int getAge() {
12.             return age;
13.         }
14.         public void setAge(int age) {
15.             this.age = age;
16.         }
17.         @Override
18.         public String toString() {
19.             return name + “ is “ + age + “ years old.”;
20.         }
21.     }
```

清单2中惟一的非样板代码是改写的 toString() 方法。构造函数和所有方法都由 IDE 生成。相比快速生成代码，在以后轻松理解它更为重要。无用的语法增加了您在理解底层含义之前必须使用的代码量。

Scala Person 类

令人震惊的是，清单 3 中用 Scala 编写的简单 3 行定义就创建了一个等效的类：

清单 3. Scala 中的等效类

```
1. class Person(val name: String, var age: Int) {
2.   override def toString = name + " is " + age
   + " years old."
3. }
```

清单3中的 Person 类浓缩成了一个可变的 age 属性、一个不可变的 name 属性，以及一个包含两个参数的构造函数，还有我改写的 toString() 方法。很容易看到这个类的独特之处，因为有趣的部分没有埋藏在语法中。

Scala 的设计强调了以最少的语法创建代码的能力，它使许多语法成为可选语法。清单 4 中的简单类演示了一个将字符串更改为大写字母的 Verbose 类：

清单 4. Verbose 类

```
1. class UpperVerbose {
2.   def upper(strings: String*) : Seq[String] = {
3.     strings.map((s:String) => s.toUpperCase())
4.   }
5. }
```

清单4中的许多代码都是可选的。清单 5 给出了相同的代码，现在使用了一个 object 而不是 class：

清单 5. 一个转换为大写的更简单的对象

```
1. object Up {
2.   def upper(strings: String*) = strings.map(_.
   toUpperCase())
3. }
```

对于等效于 Java 静态方法的 Scala 代码，可创建一个 object（与独体实例等效的 Scala 内置实体）而不是一个类。方法的返回类型、用于将单行方法主体分开的括号，以及 清单 4 中无用的 s 参数都从 清单 5 中消失了。Scala 中的这种 “可折叠语法” 有利有弊。使用可折叠语法，能够以非常符合语言习惯的方式编写代码，但这让不熟悉的人难以理解您的代码。

case 类

用作数据持有者的简单类在面向对象的系统中很常见，尤其是必须与不同系统通信的系统。这种类型的类的流行使得 Scala 项目向前推进了一步，创造了 case 类。case 类自动提供了多种便捷的语法：

可根据该类的名称创建一个工厂方法。例如，可以在不使用 new 关键字的情况下构造一个新实例：val bob = Person(“Bob”, 42)。

该类的参数列表中的所有参数都自动 val，也就是说，它们是作为不可变的内部字段来维护的。

编译器为您的类生成合理的默认 equals()、hashCode() 和 toString() 方法。

编译器将一个 copy() 方法添加到类中，以便您可返回某个副本来执行变体式更改。

Java 下一代语言不仅修复了语法瑕疵，还促进了对现代软件工作原理的更准确的理解，朝这个方向塑造它们的工具。

Groovy 的自动生成属性

在 Java 下一代语言中，Groovy 与 Java 语法最接近，为常见情形提供了称为 “语法糖 (syntactic-sugar)” 的代码生成方法。参见清单

6 中简单的 Groovy Person 类:

清单 6. Groovy Person 类

```
1. class Person {
2.   private name
3.   def age
4.   def getName() {
5.     name
6.   }
7.   @Override
8.   String toString() {
9.     "${name} is ${age} years old."
10.      }
11.    }
12.    def bob = new Person(name: "Bob",
   age:42)
13.    println(bob.name)
```

在清单6的 Groovy 代码中，定义一个字段 def 会得到一个存取函数和赋值函数。如果仅喜欢其中一个函数，可自行定义它，就像我对 name 属性所做的那样。尽管该方法名为 getName()，但我仍然可以通过更直观的 bob.name 语法访问它。

如果希望 Groovy 自动为您生成 equals() 和 hashCode() 方法对，可以向类中添加 @EqualsAndHashCode 注释。该注释使用 Groovy 的抽象语法树 (Abstract Syntax Tree, AST) 转换生成基于您的属性的方法（参阅 参考资料）。在默认情况下，此注释仅考虑属性（而不考虑字段）；如果添加了 includeFields=true 修饰符，它也会考虑字段。

Clojure 的映射式记录

可在 Clojure 中像其他语言中一样创建相同 Person 类，但这并不符合语言习惯。传统上，Clojure 等语言依靠映射（名称-值对）数据结构来持有这种类型的信息，并使用了一些处理该结构的函数。尽管仍然可以在映射中建模结构化的数据，但目前更常见的情形是使用记录。记录是 Clojure 对具有属性（常常是嵌套的）的类型名的更加正式的封装，每个实例具有相同的语义含义。（Clojure中的记录就像类 C 语言中的 struct。）

例如，请考虑以下人员定义:

```
1. (def mario {:fname "Mario"
2.              :age "18"})
```

鉴于此结构，可以通过 (get mario :age) 访问 age。简单的访问是映射上的一个常见操作。借助 Clojure，可以利用使用键充当着映射上的存取函数 的语法糖，以便使用更有效的 (:age mario) 速记法。Clojure 期望对映射进行操作，所以它提供了大量语法糖来简化此操作。

Clojure 还拥有访问嵌套的映射元素的语法糖，如清单 7 所示:

清单 7. Clojure 的速记式访问

```
1. (def hal {:fname "hal"
2.           :age "17"
3.           :address {:street "Enfield Tennis
   Academy"
4.                     :city "Boston"
5.                     :state "MA"}})
6. (println (:fname hal))
7. (println (:city (:address hal)))
```

```
8. (println (-> hal :address :city))
```

在 清单7中，我定义了一个名为 hal 的嵌套数据结构。对外部元素的访问按预期进行（(:fname hal))。如 清单 7 中倒数第二行所示，Lisp 语法执行 “内外” 评估。首先，必须从 hal 获取 address 记录，然后访问 city 字段。因为 “内外” 评估是一种常见用法，所以 Clojure 提供了一个特殊运算符（-> thread 运算符）来反转表达式，使它们更加自然、更具可读性：(-> hal :address :city)。

可使用记录创建等效的结构，如清单 8 所示：

清单 8. 使用记录创建结构

```
1. (defrecord Person [fname lname address])
2. (defrecord Address [street city state])
3. (def don (Person. “Don” “Gately”
4.          (Address. “Ennet House”
   “Boston”, “MA”)))
5. (println (:fname don))
6. (println (-> don :address :city))
```

在 清单 8 中，我使用 defrecord 创建了相同的结构，得到了一种更加传统的类结构。借助 Clojure，可以通过熟悉的映射操作和方言在记录结构中实现同样便捷的访问。

Clojure 1.2 围绕常见操作的记录定义通过两个工厂函数添加了语法糖：

->类型名称, 接收字段的位置参数

->映射->类型名称, 字段值的关键字映射

使用符合语言习惯的函数，代码由 清单 8 转换成版本 清单 9.

清单 9. Clojure 的漂亮的语法糖

```
1. (def don (->Person “Don” “Gately”
2.  (->Address “Ennet House” “Boston”,
    “MA”)))
```

在许多情况下，记录比映射和扁平结构更受欢迎。首先，defrecord 创建了一个 Java 类，使它更容易在多方法定义中使用。然后，defrecord 指定更多任务，在您定义记录时启用字段验证和其他细微处理。第三，记录速度快得多，尤其在您拥有一组固定的已知键的时候。

Clojure 结合使用记录和协议来构造代码。未来的一期文章将介绍它们的关系。

结束语

与 Java 语言相比，所有 3 种 Java 下一代语言都提供了更便捷的语法。Groovy 和 Scala 使构建类和常见情形更加轻松，而 Clojure 使映射、记录和类能够无缝地互操作。所有 Java 下一代语言的一个共同主旨是消除不必要的样板代码。在下一期文章中，我将继续探讨这个主题并讨论一些异常。 ■

专 题

JavaScript成为网络霸主的“绯闻”

JavaScript正凭借新型工具与功能提升以极度夸张的速度吞噬整个世界。我们是否应该接受这一无法逆转的趋势？

还记得那些旧日往事吗？很多用户因为担心安全问题而在浏览器中禁用JavaScript。如今这样的担忧已经显得相当可笑——JavaScript已经成为统治网络世界的绝对王者。

甲骨文公司更新下一代Java ME平台路线图

借Java社区发展东风，Java ME 8规范不断前行；Java ME 8携手Java SE 8打造更为统一的嵌入式开发平台和生态体系。

物联网的兴起，对嵌入式软件提出了诸如安全性、互操作性、可扩展性以及更快速上市等新需求，新一代Java ME平台（Java Platform Micro Edition）致力于全面应对以上需求。通过提供一致性的开发环境以及适当的的Java ME平台，新一代Java ME能够被更广范地部署在无论是微小系统还是功能强大的嵌入式系统。

甲骨文公司日前宣布了Java ME 8未来的发布规划。Java SE 8规范（Java SE 8 Specification）以及其官方参考模型JDK 8将于2014年3月发布。

Java ME 8与现有的Java ME平台相比，已完成诸多更新并具备如下新特点：

◆ 与Java SE 8采用相同的Java语言及API，极大地方便了Java SE 8和Java ME 8之间的代码共享和开发能力。

◆ 支持现代Web协议，能够与设备、企业后台系统和云服务系统实现互操作性。

◆ 更为全面的应用模式，既能够在单一用途的简单设备上使用，也可以实现复杂的配置，包括远程管理和独立管理、应用和网域审计。

◆ 先进的安全功能，包括支持TLS1.2、现代加密算法和可插拔的身份验证及授权供应商。

◆ 为电源管理和一系列标准外设提供标准API。

Oracle Java ME Embedded 8将沿用Java ME 8的标准。目前，Oracle Java ME Embedded 8预览版（Oracle Java ME Embedded 8 Early Access）可为Raspberry Pi Model B (ARM11)和ST Microelectronics STM32F4DISCOVERY (ARM Cortex-M4)提供二进制运行环境。

Java ME SDK 8预览版（Java ME SDK 8 Early Access）目前可在Windows 7及其他支持硬件平台的模拟运行中为Java ME 8预览版提供应用支持。

甲骨文公司的Oracle Java ME Embedded 8以及Oracle Java ME SDK 8预览版目前已在Oracle技术网（OTN）上发布，在获取OTN开发人员许可证后即可免费用于评估和开发。

此外，Oracle Java ME Embedded 3.4现已推出。作为Oracle Java ME Embedded的升级版，它能够更好的支持高通公司的（Qualcomm Technologies）的QSC6270T芯片组，并且增加了先进的开发能力和可维护性。

甲骨文公司也为甲骨文Java平台集成计划（Oracle Java Platform Integrator program）提供了更多的详细信息，从而使其合作伙伴能够定制Oracle Java ME Embedded以及Oracle Java SE Embedded，以应用于各类型号的设备和各个细分市场。

Oracle Java Embedded产品是甲骨文物联网策略的核心组成，具体包括Java嵌入式数据管理系统、后台数据库、大数据技术、中间设备及分析技术和高频硬件，可以帮助客户将数据转化为市场洞察力。

Java ME 8

即将面世的Java ME 8 的两项Java规范要求(JSRs)现正在Java社区进程（JCP）中接受公众审核。

Java ME 为小型嵌入式设备和移动设备提供有线连接设备配置（CLDC） 8（JSR 360）。

更新CLDC 8的目标是将功能强大且灵活的Java SE 8语言功能应用到Java ME中，另外加上API和Java虚拟机技术（JVM），以实现在维持小型封装的同时，统一系统开发环境，提高业务人员的开发能力。

Java ME Embedded Profile（MEEP）8（JSR 361）能够规范Java的规模，使其实现更广范地被采用，无论是微小系统还是功能强大的嵌入式系统。

MEEP 8充分利用CLDC 8的新功能，包含一些最新的且经过改进的功能与API，如可实现软件模块化的增强的“服务功能”应用平台，以及支持特定配置的、更为灵活的安全模型。

甲骨文Java平台集成计划

甲骨文Java平台集成计划为这些提供设备内置软件或服务、系统集成或嵌入式市场增值服务的公司而设计。该计划允许其合作伙伴移植或集成甲骨文公司的Java嵌入式产品代码，以满足其特定设备和市场需求。

该计划为参与者提供如下两方面支持：

- 配置前，提供Java平台集成开发支持，包括访问产品代码和相关TCKs，后者与甲骨文公司的工程、培训和发展支持相关。
- 配置后，提供甲骨文技术支持服务。

甲骨文Java 嵌入式合作伙伴升级

飞思卡尔（Freescale）与甲骨文公司正共同创建一个标准服务提供商平台，使得Oracle Java SE Embedded和Oracle Event Processing for Oracle Java Embedded能够与飞思卡尔的Kinetis微控制器、i.MX应用处理器或者QorIQ通信处理器结合起来。这种一体式平台有望以一种简单、统一的方式，持续提升物联网为终端用户在家庭自动化、工业和制造业自动化上所能提供的服务。具体请参阅相关新闻稿。

数字安全领域的全球领导者金雅拓（Gemalto）公司，正与甲骨文公司和智能电网系统的领先开发商V2COM合作，提供一种灵活的智能能源解决方案。该先进智能电网系统结合了金雅拓公司的Cinterion模块、甲骨文公司的Oracle Java ME Embedded、Oracle Java SE Embedded、Oracle Utilities Meter Data Management solution以及V2COM智能软件套件，希望可以实现整个北美的电力输送现代化。

高通公司（Qualcomm Technologies, Inc.）是高通集团的全资子公司，其与甲骨文公司合作，将甲骨文Java ME Embedded结合到高通公司“万物互联”（Internet of Everything，IoE）的核心芯片组中。Oracle Java ME Embedded目前在QSC6270-Turbo上已可使用，同时，甲骨文公司与高通公司也正一同努力将之拓展到MDM6x00、MDM9x15以及其他芯片组中。以上这些芯片组是世界上最广泛地被商业化使用的3G和4G芯片组。之后，Java ME 8将会成为兼容高通公司技术平台的第一代发行版。

在近日举行的圣地亚哥举行Uplinq会议上，高通公司和甲骨文公司联合展示了甲骨文公司的Oracle Java ME Embedded。APX实验室的软件工程师安德鲁·苏轧亚（Andrew Sugaya）借助高通公司的Etherios所开发IoE套件和甲骨文公司

的Oracle Java ME Embedded最终开发出了自己的LiteSense应用，该应用能够在真实世界中实现灯光的自动控制，最终获得Uplinq的黑客马拉松大奖。

Java相关资料

- 拥有遍布世界各地的9百万Java开发者
- 超过30亿的设备使用了Java技术
- 排名第1的编程语言（TIOBE编程社区指数）
- 80%的手机软件工程师使用Java平台
- 超过1.25亿的媒体设备采用了Java平台
- Java卡自面世以来，发货量已超过80亿

甲骨文高管引言

“通过理由Java的跨平台优势，Oracle Java Embeded能够轻松实现应用程序的可移植性，同时提高硬件灵活性，拓宽平台选择，并且延长产品的生命周期。”甲骨文公司Java平台开发副总裁Nandini Ramani说道，“我们很高兴的看到Oracle Java ME 8在JCP之下的规范所取得的成就，并且希望通过其与Java SE 8的校准，能够建立一个更加统一的Java嵌入式开发平台和生态系统，以更好地解决复杂的物联网相关的问题。”■

“你最喜欢的程序员漫画”精选--需求调用

2014年八大最热门IT技能

□ 核子可乐译

根据Computerworld网站组织的年度预测调查，众多IT专业人士在2014年所面临的整体就业形势与今年基本持平——今年有33%的企业有计划增加IT部门的员工数量，而未来一年则有32%的企业有此打算。

不过，虽然整体需求将继续保持稳定，但具体情况仍会出现少许变化——招聘管理者们明年将调整自己对求职者技能的关注取向。根据技术资源供应商Mondo公司创始人兼CEO Michael Kirven的说法，“对于那些拥有热门技能储备的人才而言，失业的可能性几乎为零。”他同时表示，“作为招聘者，如果您看中了某位人才，请别怀疑——至少还有其它两家企业也同时盯上了对方的才干。”

1.编程/应用程序开发

“49%的受访者声称自己有计划在未来十二个月内招揽拥有此类技能的人才。”

“去年排名：第一位。”

与2013年的预测调查一样，今年编程/应用程序开发再次蝉联热门技能排行榜冠军——不过在今年的221位受访者中，只有不到一半表示打算雇佣此类人才，远低于去年的六成。Dice控股集团（IT求职网站Dice.com的持有者）CEO Scot Melland表示，软件开发人员仍然是最抢手的技术人才，并指出这类人员的失业率也最低——根据美国劳工统计局公布的数字，其失业率仅为1.8%。有鉴于此，难怪Computerworld 2014预测调查将开发人员及程序员岗位列为未来一年中最难以弥补的空缺。Melland同时声称，在此类人才当中，又以具备移动开发专业知识与经验以及了解如何构建安全应用技能的员工最为稀缺。

在线备份服务供应商Carbonite公司希望能将业务重心转移到小型企业领域，但在转型过程中他们感受到了招聘软件开发人员及工程师的极高难度，公司人力部门副总裁Randy Bogue表示。“尽管在我们公司所在的波士顿地区，拥有丰富软件开发经验的人才并不少见，但想要雇佣他们的技术企业同样数量众多，”他解释称。“我们在寻找前端开发人员、用户体验工程师、移

动开发人员以及其它软件开发人才的时候深刻感受到了这一点。”

纽约梅隆银行CIO Lucille Mayer同样对招徕开发人才感到压力很大。这家金融服务公司的主要业务范围位于纽约及匹兹堡地区，其数百位员工当中有约四成负责开发工作。另外三成打理基础设施、两成属于业务分析及项目管理岗位，最后一成则是企业管理层。

“拥有三到五年实践经验及服务交付阅历的开发人员已经成为市场上炙手可热的稀缺资源，”Mayer指出，她个人对于拥有面向对象开发经验的人才很感兴趣。同样重要的是，企业需要吸纳具备不同背景、抱有不同想法及观点的员工。

服务业巨头凯悦集团希望摆脱以往过度依赖第三方服务供应商的窘境，这就要求他们招聘更多内部开发人才。“我们希望招揽到熟悉敏捷性、有能力将思路快速转化为原型方案并快速投付生产的人才，”凯悦集团全球技术负责人Alex Zoghlin指出。

2. 服务台/技术支持

“37%的受访者表示有计划在未来十二个月内招聘拥有此类技能的人才。”

“去年排名：第三位。”

服务台/技术支持仍然名列排行榜的三甲位置，而且由去年的第三位上升到今年的第二位。Melland表示，这样的迹象令人鼓舞，意味着经济及整体就业前景正迎来复苏。“企业之所以扩大服务台与技术支持团队的规模，是因为其员工数量有所增加、技术基础设施迎来拓展，”他解释称。此外，许多企业在尝试外包之后，又重新将技术支持工作转化为内部事务；部分原因在于移动设备的广泛普及以及企业越来越多地提供Web服务。Melland指出，由于这类机制的复杂特性，“支持人员必须真正理解企业的业务流程，从而真正让业务功能尽可能与员工的家庭环境相对接。”

经历了数年技术支持压缩，位于密歇根州迪尔伯恩市的Wolverine Advanced Materials公司计划招聘几位服务台工作人员，旨在适应公司的规模拓展以及提供基于ITIL（即信息技术基础设施库）的服务管理方案，这家汽车原材料供应商网络业务经理James Bland表示。“企业正持续发展，所以我们有信心扩大员工规模，”他表示。

3. 网络

“31%的受访者表示有计划在未来十二个月内招聘拥有此类技能的人才。”

“去年排名：第八位。”

对网络技能的需求由去年的第八位跃升至今年的第三位。根据IT招聘企业Robert Half技术公司的调查，55%的受访者将网络管理员作为最受关注技能。

对无线连接技术的需求可能是网络专业人士走红的主要原因，Melland指出。“企业对于具备无线网络人才的需求每年都在以9%的幅度增长，”他解释称，而且网络与系统管理员的失业率低至惊人的1.1%。

乔治亚州中部医疗中心首席网络分析师Charles Whitby指出，无线医疗设备的持续增加给他带来了显著的工作负担。除此之外，无线设备还带来更多网络流量，并需要组织为其提供故障排查服务——举例来说，当无线设备固件需要升级，但这类工作

并未受到美国食品及药物管理局的批准，他表示。

与此同时，在Wolverine公司，Bland正努力转移一部分网络管理职责，从而为自己节约一部分精力、转而关注于其它更具战略意义的事务。

4.移动应用与设备管理

“27%的受访者表示有计划在未来十二个月内招聘拥有此类技能的人才。”

“去年排名：第九位。”

随着移动设备在企业环境及消费者领域的广泛普及，我们对于移动技能的步步高升毫不意外——由去年的第九位达到今年的第四位。由于移动技术相对属于新兴领域，我们也能够理解Computerworld调查的受访者们将移动人才视为最难招聘对象榜上的第三位——仅次于开发及商务智能/分析技能。

在未来十二个月中，受访者们认为哪类IT员工最难于招徕?

32%编程/应用程序开发 32%

商务智能/分析 21%

移动应用程序与设备管理 17%

项目管理 14%

安全 14%

来源：Computerworld网站预测调查；受访群体：221位IT高管；2013年6月。

移动应用开发对于位于达拉斯的PrimeLending公司来讲是一项“重大举措”，公司CIO Tim Elkins表示，这也将成为明年企业的主要招聘方针。另外，为了提升自身在Salesforce.com领域的开发水平，这家抵押服务供应商希望雇佣两到三位移动开发人员。PrimeLending公司的第一款移动应用旨在帮助其业务合作伙伴——房地产经纪人与建筑商——随时查看自身贷款状态；接下来公司打算将支持对象扩展到普通消费者领域。

Elkins预计移动开发人员恐怕难以招徕，因此企业应该马上着手进行员工培训以满足需求。“Salesforce.com开发人员真的很难找到，这是因为市场对移动开发者的需求太过旺盛，”他解释称。

移动专业知识对于凯悦集团同样属于值得优先考虑的对象，Zoghlin表示该公司正尝试吸纳利基角色，从而确保自身有能力在移动及用户体验等领域保持策略的一致性。

5. 项目管理

“25%的受访者表示有计划在未来十二个月内招聘拥有此类技能的人才。”

“去年排名：第二位。”

虽然项目管理技能由去年的第二位跌落到如今的第五位，但我们仍然应该将其视为最热门的素养之一。Melland表示，Dice公司在运营中发现，项目管理人才仅次于软件开发/工程师、排在紧俏岗位榜的第二名，与去年相比支持率上升了11%。在他看来，这种上场同样应被视为整体经济迎来复苏的积极迹象，因为这表明企业乐于推动战略性项目。

Mondo公司的Kirven将市场对项目管理者的旺盛需求归结为企业在极具复杂性及战略性的业务-技术项目中表现出极大兴趣。“从历史角度看，IT部门的水平向来根据项目的成功或者失败

来判断，因此企业必须要在业务分析/项目管理层投入巨额资金，”他指出。“这类人才需要有能力与开发人员探讨技术问题及解决方案，同时又需要与业务部门交流以了解实际需求及任务优先级，从而为IT部门提供可操作的开发目标。”

6. 数据库管理员

“24%的受访者表示有计划在未来十二个月内招聘拥有此类技能的人才。”

“去年排名：未上榜。”

作为去年甚至没能上榜的IT岗位，数据库管理员将在明年爆发出巨大的发展潜力，这很可能是迎合了大数据兴起的风潮。Kirven坦言，从长远角度看，任何一家希望从保存在内部系统当中的新兴信息身上发掘价值的企业，都将不可避免地迎来大数据技术。收集信息的具体来源包括社交媒体网站、网页以及第三方公司等等。对于大数据技术的兴趣主要来自市场营销部门，相关人员希望尽可能多地了解与客户相关的信息。

“甲骨文数据库管理员以及数据架构师——这类人才只要在求职市场上冒头，一个小时之内就会被企业拉走，”Kirven指出。“企业迫切需要那些能够将数据以逻辑方式同系统相映射的人才，从而帮助业务部门汇总出与系统相关的分析及报告结论。”

能够将IT基础设施组件迁移到云环境当中的老鸟级数据库管理员将受到热烈追捧，Melland表示。他同时指出，对云技能的需求与去年相比上升了32%。

为了帮助PrimeLending公司搞定大数据项目，Elkins表示他正在积极寻找系统分析师、开发人员以及数据库管理员，从而集成来自第三方的数据、最终实现简化抵押贷款监控流程的目标。“抵押贷款长久以来似乎成为一种黑洞，极度缺乏透明度而且需要用户花费大量时间等待查询结果，”Elkins解释道。“我们的目标是在未来一年为消费者带来更多前所未有的控制能力与体验，从而改善抵押贷款的群众口碑。”

7.安全合规性/治理

“21%的受访者表示有计划在未来十二个月内招聘拥有此类技能的人才。”

“去年排名：第四位。”

安全专业知识已经是每一份热门IT技能榜单上的常客，而且Melland表示企业用户对网络安全的愈发重视将进一步推动市场对人才的需求——与去年相比需求量增长了23%。“安全技能涵盖大量具体的工作岗位，例如网络工程、软件开发以及数据库架构等，”他表示。在由Robert Half技术公司组织的调查当中，受访者们将安全工作认为最难于填补的岗位空缺之一，其它两项为应用程序开发与数据库管理。

随着恶意软件及网络攻击的日益猖獗，安全事务已经成为PrimeLending公司的头号关注重点。今年该公司已经将安全团队的规模翻了一倍——从四位成员增加到八位，Elkins告诉我们。

8.商务智能/分析

“18%的受访者表示有计划在未来十二个月内招聘拥有此类技能的人才。”

“去年排名：第五位。”

根据IDC公司的预测，从2009年到2020年，全球数据总量将扩大至最初规模的44倍，达到35.2泽字节。有鉴于此，企业迫切需要利用先进

的分析能力获得竞争优势。虽然与其它技能相比，Dice.com上商务智能/分析所提供的工作岗位仍相对较少，但Melland指出这一技能的需求增长速度在整个网站上排名第三，且与去年相比市场需求量增长了100%。分析类专业知识极度稀缺，而且在Computerworld网站的调查中位列最难雇佣人才榜的第二名。因此，相关专业人才往往薪酬惊人——一般能够达到六位数，Melland指出。

在Wolverine公司，由数据驱动的分析工作正对管理机制提出愈发强烈的需求，因此Bland希望找到既掌握商务智能技术、又熟悉该公司所使用的Plex Systems ERP应用的人才。“我们当然希望能从自己的ERP系统中获取更多信息，因此任何拥有商务智能实践经验的人才都会受到热烈欢迎，”他表示。“我们希望拥有更多更具时效性的信息，从而让企业在运营中更具主动性。”凯悦集团的Zoghlin指出，他也同样在寻找那些“能够通过分析为客户及同事带来有价值结论的人才。”

展望2014年

展望未来市场对IT员工提出的技能需求

技术类技能并非评估IT求职者时的惟一考量因素。用人单位还应当考虑申请人的人际沟通技术，从而确保新员工能顺利融入新的工作环境。根据Computerworld预测调查的结果，我们发现新员工最重要的两大特质在于协作能力（受到66%受访者的重视）以及与业务部门沟通的能力（受到62%受访者的重视）。这一结果对于Dice控制集团CEO Scot Melland而言完全在意料之中。“目前企业的每一个运营环节中都充斥着科技要素，因此我们需要雇佣擅长沟通的员工打理这些事务，”他解释道。

Wolverine Advanced Materials公司网络经理James Bland这样描述他希望在新员工身上找到的技能。“我希望新员工能帮助用户理解IT如何帮助他们以更高效的方式处理日常业务，”他解释道，而只有IT人员有能力将系统功能翻译成业务用户能够理解并顺利使用的语言后方可实现。“大家完全可以部署世界上最出色的系统，但如果用户不知道如何加以使用，那么这些系统将一文不值，”Bland总结道。

纽约梅隆银行CIO Lucille Mayer则表示，新任IT员工必须具备客户服务心态。“我们的IT部门被称为‘客户技术解决方案’，其中每位成员都对应着某一位客户，包括内部或外部客户，”她解释称。“以服务为主导、以客户为中心，协作再加上出色的沟通能力，这就是杰出IT员工必不可少的特质。”

所谓关键性沟通技能，是指利用业务部门（例如营销、销售以及财务部门）内广泛使用的语言阐释问题，Melland表示。事实上，根据Modis公司Michael Kirven的说法，除了高科技专业知识之外、用人单位也在越来越多地对IT员工的业务积累提出要求——例如某位HTML 5开发人员是否了解零售供应链，或者Java开发人员是否拥有打理金融衍生交易系统的经验。“专业化是推动创新活动的必要前提，”他解释道。

在PreimeLending公司，关键在于将所有文化加以契合。“我们将文化理念作为招聘工作的第一步，”公司CIO Tim Elkins指出。这种处理方式在雇佣企业高层管理者时尤为重要。“如果我们打算聘用一位新经理，那么不止要看他是否是一位出色的领导者、更要观察他能否按照我们的风格融入工作，”Elkins将他们的要求称为“公仆式领导”——也就是说领导者要抱着为他人服务的心态、而不能仅仅对普通员工颐指气使。■

在本文中，我们将论证与当前“最新最强”技术方案相关的七种实验流程。请放心，这些技术都已经具备一定发展历程——事实上，其中一些已经有超过二十年的历史。但它们仍然能够为现代企业堆栈提供显著的领先优势，也因此受到技术行业的广泛关注。

值得尝试的七大前沿性编程实验

Erlang、Node.js、Go：本文将指导大家迈出走向热门编程语言的第一步。

“前沿”这个字眼可能太过激烈，甚至会直接吓跑一些保守的企业技术人员。对于企业IT部门而言，利用前沿性技术打造新产品简直有几分痴人说梦的味道。

这种反应不只源自对新方案的恐惧或者现有机制太过陈旧导致难以更新，毕竟企业团队需要采纳一切可资借鉴的新思路来指导业务流程--甚至包括新思路的阶段性成果。但从管理者的角度出发，他们也需要尽可能保持业务体系的稳定性，因此原有堆栈代码带来的确定性能带来理想的使用安全感。

使用前沿性技术的关键在于多做实验而不能粗暴地组织全面升级。在预期效果最明显的领域尝试部署新代码，观察这些前沿工具能否提供我们所需要的性能及功能，然后权衡业务流程的基础要素是否受到了影响。单靠实验并不一定能立即暴露出新技术中的全部潜在弱点。因此逐步增加对应工作量，在审查结果达到全面准确之后再以严谨的态度进行实施才是正途。

在本文中，我们将论证与当前“最新最强”技术方案相关的七种实验流程。请放心，这些技术都已经具备一定发展历程——事实上，其中一些已经有超过二十年的历史。但它们仍然能够为现代企业堆栈提供显著的领先优势，也因此受到技术行业的广泛关注。不要抗拒，请以积极的心态加以尝试。这些技术是大家从熟悉的世界迈向新纪元的理想起点，更快、更简单、更纯粹——不容错过。

前沿实验第一位：利用Erlang实现纯粹的并发效果

目前前沿技术的主要诉求在于解决实际问题——换言之就是治标不治本。云计算之所以能够兴起，是因为数据中心的管理工作令IT部门十分头痛。而像Erlang这样的新语言之所以不断涌现，则是因为技术老鸟们无法胜任新时代下的技术要求。

作为一种拥有二十多年历史的编程语言，Erlang凭借着自身的高效结构将并发线程的执行难度降至最低，从而赢得了越来越多技术人员的青睐。如果利用Erlang进行编程，Web服务器能够在多用户并发任务当中带来更出色的表现——这是由于该语言的设计初衷旨在帮助开发人员通过限制程序的编写方式来做出正确决策。当然，技术从牛们完全可以利用其它语言实现同样的功效，但Erlang在设计中采用一系列安全辅助机制，从而避

免线程紊乱状况的发生。Erlang语言为需要被锁定及解锁的共享变量中引入了功能性设计及消息传递机制，这使得IT部门所开发的企业集群能够更好地应对多用户环境。

Erlang语言由Ericcson开发，最初是为了用于内部电信系统，而后逐渐演变为开源项目。在Erlang步入开源道路之后，其大型技术社区如今已经提供大量支持工具，例如在大部分主流操作系统中进行基本运行安装，以及其它开源项目。很多参与者利用OTP——全称为开放电信平台——作为网站数据服务的交付基础，这也是大部分简单项目的最佳起点。（大家可以点击此处下载Erlang。）

不过任何一种语言都存在局限。Erlang的主要瓶颈在于，其新工具的设计目的并不是为了修复那些由失误或故障引发的问题；它们关注的是提供差异化决策方案。也就是注重“取舍”而非“问题”。

举例来说，云计算能够带来出色的简便性与灵活性，但却会同时造成控制与安全方面的难题。使用Erlang语言的程序员需要牺牲一部分自由度来换取新的开发模式。如果大家的代码需要面对的是多位彼此互不相关且无需沟通的用户，那么使用Erlang模式来编写代码是最便捷的方式。不过如果大家需要让自己的线程彼此沟通——前提是各位有能力攻克这项难关——使用Erlang反而会让事情变得更加复杂。

从小处着手，了解Erlang是否能够与自己的实际需求相吻合，而后通过取舍获得最佳代码表现，这就是我们进行实验的意义所在。

前沿实验第二位：Node.js Web堆栈

很多企业在评判服务的实际效果时，都会观察其是否能够快速交付数据。没人愿意让一位潜在客户面对着空白的浏览器窗口破口大骂。此外，在向顶头上司进行工作汇报时，大家也肯定不希望把时间浪费在等待关键性业务分析报告生成身上。

不少前沿性工具都专门为速度而生。以Node.js为例，它的流行主要是因为其出色的运行速度。它在与新型NoSQL数据库协作时的速度表现甚至更快，这是由于新型NoSQL数据库在数据保存方面速度拔群。总之，我们可以在这套小型平台上建立起高速Web基础设施，并且同时降低对电力资源的消耗。速度与能效之间往往存在着紧密联系。

这样的速度表现对于将快速响应视为首要目标的企业用户来说极具吸引力。更短的网站响应时间能够大大提高客户的第一印象，从而将潜在客户转化为实实在在的买家。不过银行等拥有大量固定客户的企业对于这方面特性可能并不关注。

Node.js是一套以Chrome V8 JavaScript引擎为基础创建的开源堆栈，但大多数技术人员会在nodejs.org网站为各类主流平台寻找预先开发好的可执行方案。作为主要赞助商，Joyent公司还提供配备镜像的云设备，包含所有必要库及工具。

很多开发人员习惯于直接向Web框架求助，例如Tower、Geddy或者Railway，它们能够切实简化数据驱动的基础网站的开发流程。

Node.js的局限与性能无关，最大的问题在于它给开发人员带来沉重的技术负担。即使是最为睿智的程序员也需要加倍小心，因为这意味着所有数据包都运行在同一进程当中。如果某位用户的无意操作偶然触发了代码中的bug，整个Web服务器都将陷入崩溃。优秀的程序员以及严格的测试流程能够避免问题的发生，但没人能保证永远不出纰漏。在这方面，Erlang的做法正好相反——它利

用多项限制帮助程序远离严重错误。

Node.js与NoSQL相结合完全能够成为当下前沿实验的理想方向之一：专注于为爆炸式发展中的社交网络提供支持。如果大家打算亲自进行实验，请务必选择对速度要求较高、但对稳定性不太关注的领域。如果您的数据需要精心打理，请告别Node.js、远离风险。

前沿实验第三位：HTML 5 Web与移动应用

古语有云“新官上任三把火”，刚刚诞生的新工具也值得我们体验一下其旺盛火力。从零开始刚刚建立完成的最新语言及软件堆栈很可能尚未经过新版本带来的细微调整并舍弃不合用的API，但其语法及格式也因此比较纯粹、简洁。

这往往能为大家带来更简单、更纯粹的代码。尽管程序员们能够利用任何语言编写出复杂的代码，但新型堆栈往往不需要经过太多代码修复及版本测试流程。智能手机上的某些应用需要经过几十个版本的严格测试，从而确保其能够以正确版本发挥正确效果。新型堆栈就不至于带来这样的额外复杂性。

目前市场上已经存在多种HTML 5项目，旨在为开发者提供创建网站或者移动手机应用所必需的基本要素。代码，或者被称为框架或者支架，能够在页面中组织内容并提供由菜单为主导的过渡机制。其中最具人气的项目包括jQuery Mobile、Sencha Touch以及Titanium，但近来还兴起了更多其它工具。很多人气极高的CMS堆栈，例如WordPress或者Drupal，已经开始将关注重点转移到移动环境当中，而且往往能够使用大量原始代码。

虽然这些新的代码堆栈足够简洁，但它们需要将原有平台彻底淘汰之后才能实现效果。新工具能够轻松帮助开发者编写出简单而精致的代码。它们往往直接忽略掉陈旧硬件以及操作系统版本。当然，它们的简洁性与高速性源自对现有预发布代码的高度依赖。

HTML 5框架往往会在大家使用旧版本或者不符合标准的浏览器时发生故障。突然之间，菜单显示的位置出现严重偏移，而且文字内容也开始只显示一半——这说明CSS指令无法正常工作。有时候新需求需要与旧方案和谐相处，但新代码却坚持以同一种方式实现某一种功能，这也是最令人头痛的问题。

在迈入实验阶段之前，请认真考虑自己是否有能力为技术的特定子集提供必要支持。

前沿实验第四位：利用R语言进行数据处理

从简洁Web设计到更为复杂的大数据分析，R语言已经成为目前大部分热门新工具的开发核心——这些成果往往被用于通过数据解决问题或者掌握客户情况。在众多成果工具集的支持下，R不仅是一种能够为通用统计公式提供预定义功能的语言，更是一种思考问题并找寻解决方案的全新方式。

举例来说，大数据分析软件包中的统计模型能够识别并标记复杂的模式并充分发挥现代计算机集群所提供的全部性能资源。统计模型取代了原本只能排序或者寻找最大值的简陋机制。与前沿统计软件协作意味着大家能够实现深层次分析，并在旧有代码无能为力的状况下找到有价值的蛛丝马迹。

这些新型视角的出现帮助企业节约了数十亿美元的常规支出。他们帮助店铺确定所在地域的口味喜好，根据人们的审美习惯以颜色、图案、大小等标准安排货架摆放方式。它们汇总出的结

论帮助营销人员精准确定广告投放量。总之，只要有数据的地方，我们就有机会从中找到有利于自身的发展机遇。

作为开源项目，R语言专注于通过培养机制建立核心用户群体。很多开发人员会以R Studio等更为完整的IDE作为起点，因为这些IDE捆绑有编辑器以及具备执行引擎的输出窗口。在生产堆栈领域，R Studio IDE已经成为最理想的开发机制。

不过像R语言这样的统计工具也有缺陷，其结果并非永远直观，实验所获得的效用也常常不够明显。这是因为尽管思路足够前卫，但其执行流程还不够科学。大数据分析是一套极为出色的理论，甚至堪称伟大的灵感，但几乎没人能准确说出这项技术到底有多出色——特别是在背景条件的影响之下。统计分析真能帮助大家改善自己的产品吗？收集到的数据能否带来理想的精确度以指导工作？没人说得清，但如果能花上几个月组织实验，大家没准会得到自己的结论。

考虑到R语言等统计工具令人激动的特性，我们不禁急于利用它对磁盘阵列中保存的数据进行一番全面分析。也许大家运气很好，巨大的机遇正在磁盘中静静等着你来发现。不过很多技术人员已发现，单靠大数据分析机制还不足以彻底实现“去其糟粕、取其精华”的目标，人力的介入不可避免。单靠分析结果中那一串串数字只会让人找不着北。

前沿实验第五位：体验NoSQL的极致速度

让我们面对现实：我们程序员其实是个相当懒散的群体。我们不愿意从零开始创建项目——除非不得不做。新工具的出现往往是由于对新型功能的强烈需求。有时候情况甚至更为严重。

获得这些新功能的惟一途径就是接受新型工具。许多新型NoSQL数据库能够毫不费力地迁移到云环境当中。弄一堆设备，在它们之间顺利运作，这正是技术人员的专长与构建基础设施的出发点。总之，如果没必要，IT部门根本不会有热情引进新机制。

目前可供选择的NoSQL数据库可谓层出不穷，其中大部分项目在功能方面存在广泛交集。对这些交集进行一一列举并详加解释是项巨大的工程，受篇幅所限，我在这里就不再赘述了。总之，目前比较热门的工具有Cassandra、MongoDB、CouchDB以及Riak。某些企业还会提供工具即服务方案。MongoLab以及MongHQ就是两套利用MongoDB实现数据存储的方案，只要版本版本相近即可实现兼容。

以闪电般的速度以及灵活的扩展性实现响应非常重要，为了充分享受新工具在这两方面带来的提升，大家值得对手中的全部现有代码进行重写。不过此类前沿方案的核心吸引力还在于，我们找不出它们在发展过程中误入歧途的端倪。通常情况下，技术方案往往存在阴暗面，我们需要通过艰辛的探索——甚至错误——才能全面地对其加以认识。

NoSQL数据库也面临着同样的问题。它的速度确实够快，但这主要是因为它并不提供任何坚实的一致性承诺。这类数据库项目单纯接纳大量数据并在确定全部数据都已经写入磁盘之前就显示“全部完成”信息。对于社交网站这类内容不太重要的企业来说，个人用户状态信息的丢失不至于惹出什么大麻烦，但其它企业的心态可就没这么轻松了。

寻找合适的区域，确保其中不涉及任何关键性数据，接下来就可以放心大胆地鼓捣这些键-值数据存储方案了。

前沿实验第六位：利用图形数据库寻找连接

数据库概念确立于上个世纪。简单来说，我们首先定义包含着特定数据列的列表，然后向其中插入行，全部填满后就形成了一套数据库。数列中可以包含整数、十进制数字或者字符串，传统数据库的灵活性也就仅限于此了。

但Neo4j等图形数据库的出现给数据库概念引入了新思路。我们仍然可以在数列中添加数字及字母，但现在大家还可以在不同单元行之间创建指针以形成网络结构。如果存储内容是社区网络，那么数据库就能记录下每位用户以及与之相关的好友。

在规则数据库中，我们可以为每个单元行赋予一个键，并将所有指针以键的形式保存在同一个列当中。图形数据库的强大实力体现在用户运行查询之时。图形数据库能够对网络进行解压，并利用经过精密调整的搜索算法组织网络查询。它不需要像关系数据库那样处理复杂的链接与加入关系。如果大家希望查询某位用户的朋友的朋友的朋友一共有多少位，查询引擎能够直接给出结果。如果大家希望测试两位用户之间需要经过多少次朋友关系跳转才能彼此联系，引擎则需要搜索网络并找出答案。

Neo4J由Neo技术公司所打造，提供三种版本并遵循多种许可模式。社区版以GPL 3.0许可为基础并提供所有搜索能力。高级版及企业版则增加了多种工具，用于监控数据吞吐量、实现集群同步并对数据库进行备份。二者都遵循针对实验及开源项目的Affero GPL许可，能够支持且无需公开大家自己的代码。

在功能集方面，我们同样需要做出取舍。图形数据库与其它数据库家族成员相比，在开发水平及调整精度上有所欠缺。它们在图形算法领域可算当之无愧的专家，但在传统功能方面则既不够渊、也不够博。选择图形数据库就意味着放弃其它功能取向。

前沿实验第七位：利用Go简化结构

多年以来，各类编程语言就像雨后春笋一般争相涌现。由于每个人都希望把自己喜爱的功能及思路添加进来，因此很多原本单纯的想法最终变成了一大坨重量级负担。现在是时候创建一套崭新而又经过修剪的王牌语言了。

Go就是这样一款由谷歌公司的众多技术专家创建出的语言。其语法机制对于熟悉C及Java的程序员来说并不陌生，而且其本身难度也绝对亲民。我们不仅可以通过定义类型对代码进行编译，甚至还能在代码运行过程中对其加以修改。无用存储单元收集程序负责所有内存分配任务。Go还提供一套轻量级机制用于组织并发方法，这样大家就能轻松编写出支持并行运行的代码了。

谷歌以自由度极高的开源许可为基础，为Unix、Linux、Mac OS X以及Windows等系统平台提供了编译器与运行堆栈。目前已经有多家企业开始以实验方式测试Go语言，而谷歌也表示尝试将其代码引入某些生产环境。大家可以访问tour.golang.org以交互方式了解这款新语言。

Go这样的语言最适合帮助企业用户在重新组织或者设计办公环境时清理原有负担。对结构加以精简能够简化员工之间的协作难度，因为大家的沟通将变得更顺畅、也能够更轻松地向预定目标共同努力。Go的支持者们对其特性大加赞美，认为它能够帮助自己与他人协作开发出简洁而极具功能性的产品。简洁性将协作过程中经常出现的沟通与同步障碍一扫而空，这也正是Go语言的核心价值所在。■

Indexed DB入门导学

□ 廖煜嵘译

在html 5中,其中一个引人注意的新特性,那就是允许使用Indexed DB。用户可以从这个链接（http://www.w3.org/TR/IndexedDB/）了解到Indexed DB的详细标准。在本文中，将对Indexed DB作简单的入门介绍。

概述

从本质上说，IndexedDB允许用户在浏览器中保存大量的数据。任何需要发送大量数据的应用都可以得益于这个特性，可以把数据存储在用户的浏览器端。当前这只是IndexedDB的其中一项功能，IndexedDB也提供了强大的基于索引的搜索api功能以获得用户所需要的数据。

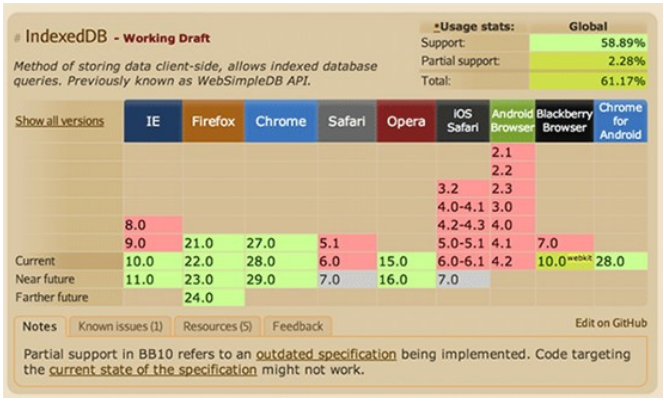
用户可能会问：IndexedDB是和其他以前的存储机制(如cookie,session)有什么不同?

Cookies是最常用的浏览器端保存数据的机制,但其保存数据的大小有限制并且有隐私问题。Cookies并且会在每个请求中来回发送数据,完全没办法发挥客户端数据存储的优势。

再来看下Local Storage本地存储机制的特点。Local Storage在HTML 5中有不错的支持,但就总的存储量而言依然是有所限制的。Local Storage并不提供真正的“检索API”,本地存储的数据只是通过键值对去访问。Local Storage对于一些特定的需要存储数据的场景是很适合的，例如，用户的喜好习惯，而IndexedDB则更适合存储如广告等数据（它更象一个真正的数据库）。

在我们进一步探讨Indexed DB前，我们先看下目前的主流浏览器对Indexed DB的支持。IndexedDB目前依然是一个w3c中候选的建议规范，在这一点上规范目前还是令人感到满意的，但现在正在寻找开发者社区的反馈。该规范可能会在到最后确认阶段前会因应w3c的建议有所变化。在一般情况下，目前的浏览器对IndexedDB的支持都以比较统一的方式实现，但开发者应注意在未来的更新及对此作出一定的修改。

我们来看来自CanIUse.com的对于各浏览器对IndexedDB的支持的图表，可以看到，目前桌面端浏览器对其的支持是不错的，但移动端浏览器的支持就比较少了：



Chrome for Android支持IndexedDB,但很少人目前在Android设备上使用这款浏览器。是否缺乏移动端浏览器的支持就意味着不应该使用它呢?当然不是！幸好我们的开发者都懂得持续改进的概念。象IndexedDB这样的特性可以用其他的方式添加到那些不支持该功能的浏览器中。用户可以使用

包装过的类库去转换到移动端的WebSQL，又或者干脆不在移动端进行本地存储数据。我个人认为能在客户端缓存大量的数据，对使用上来说是很重要的功能，即使缺乏移动端的支持。

开始学习

首先，在使用IndexedDB前，要做的是检查当前的浏览器对IndexedDB是否支持，做法只需要使用如下代码就可以实现：

```
1.  if(“indexedDB” in window) {
2.      console.log(“YES!!! I CAN DO IT!!! WOOT!!!”);
3.  } else {
4.      console.log(“I has a sad.”);
5.  }
6. },false);
```

上面的代码中（可以在本文下载代码中的test1.html中找到），使用了DOMContentLoaded事件在加载的过程中，通过判断在window对象中是否存在indexedDB，当然为了在接下来的过程中记住判断的最终结果，可以使用如下的代码更好地保存（test2.html）：

```
1. var idbSupported = false;
2. document.addEventListener(“DOMContentLoaded”, function(){
3.  if(“indexedDB” in window) {
4.      idbSupported = true;
5.  }
6. },false);
```

操作数据库

下面要讲解的是如何操作IndexedDB数据库。

首先要了解的是，IndexedDB并不象传统的如SQL Server那样需要额外安装。Indexed是存在于浏览器端的并且能被用户所访问控制。IndexedDB和cookies和local storage的原则是一样的，就是一个IndexedDB是和一个唯一的DOMAIN相关联的。比如名为“Foo”的数据库是由foo.com所关联的，是不会和goo.com所创建的同名“Foo”数据库冲突的，因为他们属于不同的domain，并且他们之间是不能互相访问的。

打开一个数据库是通过命令执行的。基本的用法是提供数据库的名称和版本号即可，其中版本是十分重要的，稍候会作解析。下面是基本的例子：

```
var openRequest = indexedDB.open(“test”,1);
```

打开一个IndexedDB数据库是异步的操作。为了处理操作的返回结果，需要增加一些事件的监听，目前有四种不同类型的事件监听事件：

- success
- error
- upgradeneeded
- blocked

大家可能已经能知道success和error事件的含义了。而upgradeneeded事件是在首次打开数据库或者改变数据库版本的时候被触发。blocked事件是在前一个连接没有被关闭的时候被触发。

让我们看下接下来的例子(test3.html)，其中当首次访问网站的时候会触发upgradeneeded事件，然后是success事件。

```
1. var idbSupported = false;
2. var db;
```

```
3. document.addEventListener( “DOMContentLoaded”, function(){
4.
5.    if( “indexedDB” in window) {
6.        idbSupported = true;
7.    }
8.    if(idbSupported) {
9.        var openRequest = indexedDB.open( “test”,1);
10.           openRequest.onupgradeneeded = function(e) {
11.               console.log( “Upgrading...”);
12.           }
13.           openRequest.onsuccess = function(e) {
14.               console.log( “Success!”);
15.               db = e.target.result;
16.           }
17.           openRequest.onerror = function(e) {
18.               console.log( “Error”);
19.               console.dir(e);
20.           }
21.       }
22.    },false);
```

在上面的代码中，我们再一次检查当前浏览器是否支持IndexedDB，如果支持，则打开一个数据库。在这段代码中我们使用了三个事件——upgrade事件、成功事件和错误事件。首先看success事件,该事件通过target.result传入句柄，然后将其复制到一个全局变量db中，用作稍候添加数据用。如果用户在支持IndexedDB的浏览器中运行上面的代码，应该会在控制台看到看到Upgrading...和success的输出信息，如果再次运行的话，则只会看到输出成功的信息，因为upgradedneed事件只在首次打开数据库或升级的时候被调用。

对象存储

接下来，我们学习如何存储数据。IndexedDB有一个概念称为“对象存储”。

用户可以认为这是一张典型的关系数据库中的表。对象中当然会存储了数据，但也有一个keypath和可选的索引集合。所谓的Keypaths是用户数据的唯一标识，并且可以用不同的格式表示。索引则在后面会进行讲解。

还记得之前提到的upgrademeeded事件么？要注意的是只能在upgradeneeded事件中创建一个对象。目前，默认是在用户首次访问你的网站的时候会自动创建对象。同时如果要修改你的对象，则必须更新数据的版本，并且要为此编写代码，让我们来看代码的例子如下：

```
1. var idbSupported = false;
2. var db;
3. document.addEventListener( “DOMContentLoaded”, function(){
4.    if( “indexedDB” in window) {
5.        idbSupported = true;
6.    }
7.    if(idbSupported) {
8.        var openRequest = indexedDB.open( “test_v2”,1);
9.        openRequest.onupgradeneeded = function(e) {
```

```
10.             console.log(“running
  onupgradeneeded”);
11.             var thisDB = e.target.result;
12.             if(!thisDB.objectStoreNames.
  contains(“firstOS”)) {
13.                 thisDB.
  createObjectStore(“firstOS”);
14.             }
15.         }
16.         openRequest.onsuccess =
  function(e) {
17.             console.log(“Success!”);
18.             db = e.target.result;
19.         }
20.         openRequest.onerror = function(e) {
21.             console.log(“Error”);
22.             console.dir(e);
23.         }
24.     }
25. },false);
```

上面的代码（见test4.html）中，请看upgradeneeded事件，首先是通过变量thisDB获得了打开的数据库，这个变量的其中一个属性是一个已存在的对象存储的list，名为objectStoreNames。我们可以使用contains方法检查某个对象是否已经存在了，如果不存在则可以进行创建，使用的方法是createObjectStore，因为这个是IndexedDB的同步操作，因此不需要为此进行事件监听。

总结一下，当用户访问你的网站时，如果用户的浏览器支持IndexedDB，则首先触发的是upgradeneeded事件，代码中检查是否有“firstOS”对象存在，如果没有的话则新创建一个，当用户第二次访问网站的时候，数据库的版本依然和第一次访问时是相同的。

假如需要新增加另外一个对象存储，则只需要增加版本号并复制上面contains/createObjectStore部分的代码，代码如下（见test5.html）：

```
1. var openRequest = indexedDB.open(“test_
  v2”,2);
2. openRequest.onupgradeneeded = function(e) {
3.     console.log(“running
  onupgradeneeded”);
4.     var thisDB = e.target.result;
5.     if(!thisDB.objectStoreNames.
  contains(“firstOS”)) {
6.         thisDB.
  createObjectStore(“firstOS”);
7.     }
8.     if(!thisDB.objectStoreNames.
  contains(“secondOS”)) {
9.         thisDB.
  createObjectStore(“secondOS”);
10.         }
11.     }
```

如何增加数据

下面我们可以开始增加数据。和传统的关系型数据库有点不同，IndexedDB允许保存对象。这意味着开发者甚至可以保存一个Javascript对象。对于数据查询是需要使用事务，事务需要两个参数，第一个是将要处理的表的数组，其中的元素是表，

第二个参数是事务的类型。目前有两种事务的类型：只读和读写。增加数据是属于读写操作，代码如下：

```
1. var transaction = db.transaction([“people”],
   ”readwrite”);
```

这里指出了要设置存储对象people为读写操作，然后使用objectStore指定要操作的存储对象，存储到变量store中去

```
1. var store = transaction.
   objectStore(“people”);
```

接下来就可以增加数据了，我们定义一个person对象，然后增加到这个store中去：

```
1. //Define a person
2. var person = {
3.     name:name,
4.     email:email,
5.     created:new Date()
6. }
7.
8. //Perform the add
9. var request = store.add(person,1);
```

可以看到，这里声明了一个Javascript的普通对象，然后使用store的add方法就可以增加这个对象到对象存储中，其中add的第2个参数是标识数据的唯一标识，在这里只是暂时硬编码了，在接下来我们将不再使用硬编码的方法。

要注意增加数据的操作是异步的，所以增加两个事件监听，代码如下：

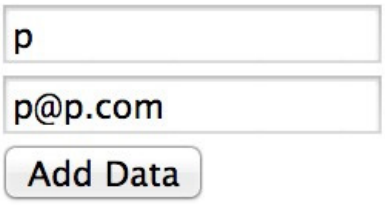
```
1. request.onerror = function(e) {
2.     console.log(“Error”,e.target.error.name);
3.     //some type of error handler
4. }
5. request.onsuccess = function(e) {
6.     console.log(“Woot! Did it”);
7. }
```

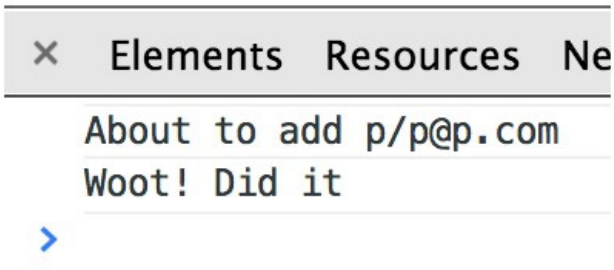
现在我们看下test6.html，这个是包含html的完整代码：

```
1. <!doctype html>
2. <html>
3. <head>
4. </head>
5. <body>
6. <script>
7. var db;
8. function indexedDBOk() {
9.     return “indexedDB” in window;
10.     }
11.     document.addEventListener(“DOMConte
   ntLoaded”, function() {
12.         //No support? Go in the corner and
   pout.
13.         if(!indexedDBOk) return;
14.         var openRequest = indexedDB.
   open(“idarticle_people”,1);
15.         openRequest.onupgradeneeded =
   function(e) {
16.             var thisDB = e.target.result;
17.             if(!thisDB.objectStoreNames.
   contains(“people”)) {
18.                 thisDB.
   createObjectStore(“people”);
```

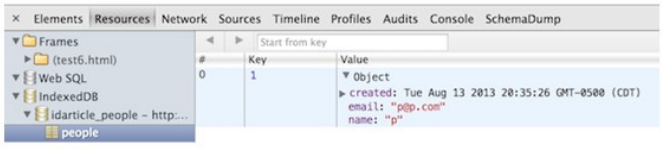
```
19.             }
20.         }
21.
22.         openRequest.onsuccess = function(e)
    {
23.             console.log("running onsuccess");
24.             db = e.target.result;
25.             //Listen for add clicks
26.             document.
    querySelector("#addButton").
    addEventListener("click", addPerson, false);
27.         }
28.         openRequest.onerror = function(e) {
29.             //Do something for the error
30.         }
31.     },false);
32.     function addPerson(e) {
33.         var name = document.
    querySelector("#name").value;
34.         var email = document.
    querySelector("#email").value;
35.         console.log("About to add
    "+name+"/"+email);
36.         var transaction = db.transaction(["pe
    ople"],"readwrite");
37.         var store = transaction.
    objectStore("people");
38.         //Define a person
39.         var person = {
40.             name:name,
41.             email:email,
42.             created:new Date()
43.         }
44.         //Perform the add
45.         var request = store.add(person,1);
46.
47.         request.onerror = function(e) {
48.             console.log("Error",e.target.error.
    name);
49.             //some type of error handler
50.         }
51.         request.onsuccess = function(e) {
52.             console.log("Woot! Did it");
53.         }
54.     }
55.     </script>
56.     <input type="text" id="name"
    placeholder="Name"><br/>
57.     <input type="email" id="email"
    placeholder="Email"><br/>
58.     <button id="addButton">Add Data</
    button>
59.     </body>
60.     </html>
```

在浏览器中运行这段代码，则可以看到如下图的HTML页面，其中在文本框输入内容，点按钮，则会在浏览器的调试工具中（我们使用的是Chrome）看到有如下的输出：

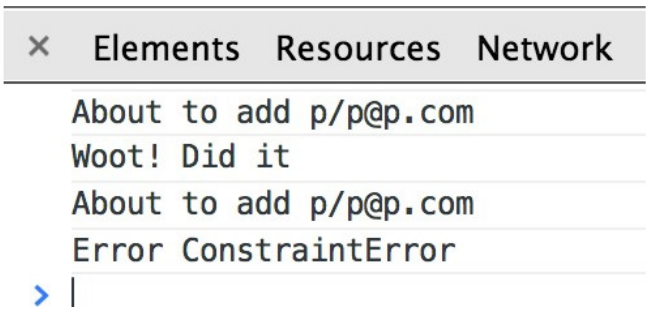




.如果使用Chrome，则可以充分利用其调试工具中对IndexedDB的可视化支持，如果点资源的TAB，展开IndexedDB的部分，则可以看到如下图所示的，我们之前创建的people存储对象。



接下来我们尝试再次点击增加按钮，这个时候注意到会输出错误的提示信息如下图，其信息表示试图增加一个已经存储的对象，违反数据约束了：



关于Key

Keys是IndexedDB的主键，传统的关系数据库可以没有keys，但每个对象存储都必须有一个key。IndexedDB允许多个不同类型的KEY。

第一种方法是象上文中的那样手工指定。第二种方法是使用keypath，其中的key是基于数据自身的属性，比如people中的例子可以使用email地址作为key。第三种方法是建议采用的方法，就是使用key生成器，这有点象自动编号的主键的方法。下面是关于key的两个例子，其中一个是使用keypath,另外的是使用key生成器：

```
1. thisDb.createObjectStore(“test”, { keyPath:
    “email” });
2. thisDb.createObjectStore(“test2”, {
   autoIncrement: true });
```

并且我们可以修改上一个例子中，用带key的方法创建对象：

```
1. thisDB.createObjectStore(“people”,
   {autoIncrement:true});
```

这样的话，就可以取消之前硬编码的做法了：

```
1. var request = store.add(person);
```

读取数据

再来看如何读取数据，读取数据要在一个事务中进行并且是异步的，下面是例子：

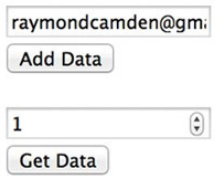
```
1. var transaction = db.transaction([“test”],
    “readonly”);
2. var objectStore = transaction.
   objectStore(“test”);
3. //x is some value
4. var ob = objectStore.get(x);
5.  ob.onsuccess = function(e) {
6. }
```

在上面的代码中，首先设置为只读的事务。然后使用objectStore.get方法就可以了，要注意编写其onsuccess事件，也可以使用链式调用，如

下：

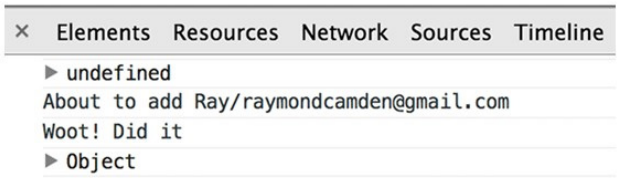
```
1. db.transaction([“test”], “readonly”).
   objectStore(“test”).get(X).onsuccess =
   function(e) {}
```

现在我们在test8.html中，为我们之前的例子加上读取数据的部分，运行后的效果如下图：



Key 1

name=Ray
email=raymondcamden@gmail.com
created=Tue Aug 13 2013 21:25:49 GMT-0500 (CDT)



其中读取数据的代码为：

```
1. function getPerson(e) {
2.     var key = document.
   querySelector(“#key”).value;
3.     if(key === “” || isNaN(key)) return;
4.     var transaction = db.transaction([“people
   ”],”readonly”);
5.     var store = transaction.
   objectStore(“people”);
6.     var request = store.get(Number(key));
7.     request.onsuccess = function(e) {
8.         var result = e.target.result;
9.         console.dir(result);
10.             if(result) {
11.                 var s = “&lt;h2>Key
   “+key+”&lt;/h2>&lt;p>”;
12.                 for(var field in result) {
13.                     s+= field+”=”+result[field]+”
   &lt;br/>”;
14.                 }
15.                 document.
   querySelector(“#status”).innerHTML = s;
16.             } else {
17.                 document.
   querySelector(“#status”).innerHTML =
   “&lt;h2>No match&lt;/h2>”;
18.             }
19.         }
20.     }
```

如何读取更多数据

如何读取批量的数据？也就是象传统关系数据库中读取数据集？IndexedDB支持游标的概念，这对有数据库基础的读者来说是很容易理解的，下面的代码演示了如何读取数据集：

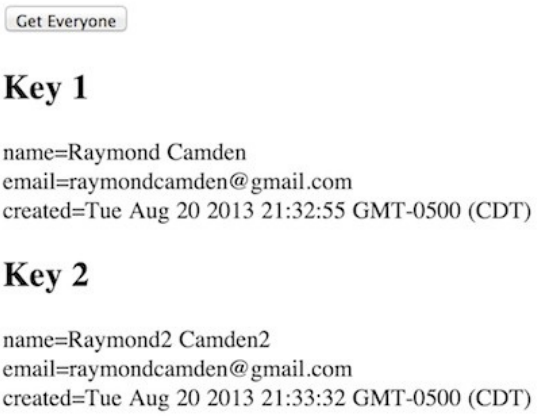
```
1. var transaction = db.transaction([“test”],
   “readonly”);
2. var objectStore = transaction.
   objectStore(“test”);
3.
4. var cursor = objectStore.openCursor();
5.
6. cursor.onsuccess = function(e) {
7.     var res = e.target.result;
8.     if(res) {
9.         console.log(“Key”, res.key);
```

```
10.            console.dir(“Data”, res.value);
11.            res.continue();
12.        }
13.    }
```

其中使用objectStore的openCursor打开游标，并且在onsuccess事件中进行处理,注意使用res获得了结果集,然后用res.continue()方法去循环读取记录集。同样，我们用这个方法去遍历之前的people存储集，代码如下：

```
1. function getPeople(e) {
2.    var s = “”;
3.    db.transaction([“people”],
   “readonly”).objectStore(“people”).
   openCursor().onsuccess = function(e) {
4.        var cursor = e.target.result;
5.        if(cursor) {
6.            s += “&lt;h2>Key “+cursor.
   key+”&lt;/h2>&lt;p>”;
7.            for(var field in cursor.value) {
8.                s+= field+”=”+cursor.
   value[field]+”&lt;br/>”;
9.            }
10.               s+=”&lt;/p>”;
11.               cursor.continue();
12.           }
13.           document.
   querySelector(“#status2”).innerHTML = s;
14.       }
15.   }
```

可以在test9.html中看到完整的代码，我们可以不断增加人员信息，然后点获取所有信息，如下图运行效果：



IndexedDB中的索引

在本教程的最后部分，讲解IndexedDB中最重要的部分就是索引了。首先是如何创建索引，这需要在upgrade事件中进行，下面是方法：

```
1. var objectStore = thisDb.
   createObjectStore(“people”,
2.            { autoIncrement:true });
3. //first arg is name of index, second is the
   path (col);
4. objectStore.
   createIndex(“name”,”name”, {unique:false});
5. objectStore.
   createIndex(“email”,”email”, {unique:true});
6. 其中，在objectStore.
   createIndex(“name”,”name”, {unique:false});
```

中，第一个参数就是索引的名称，第二个参数就是列。并且使用unique指定某个列是否唯一，这里只是认为email是唯一的，name是不唯一的。

那么如何使用索引呢？一旦我们在事务中获得对象存储，则可以通过索引去获得数据，一个例子如下：

```
1. var transaction = db.transaction([“people”],
   ”readonly”);
2. var store = transaction.
   objectStore(“people”);
3. var index = store.index(“name”);
4. //name is some value
5. var request = index.get(name);
```

接下来，可以在onsuccess事件中进行编码处理，参考test10.html，代码如下：

```
1. request.onsuccess = function(e) {
2.     var result = e.target.result;
3.     if(result) {
4.         var s = “&lt;h2>Name “+name+”&lt;/
   h2>&lt;p>”;
5.         for(var field in result) {
6.             s+= field+”=”+result[field]+”&lt;
   br/>”;
7.         }
8.         document.
   querySelector(“#status”).innerHTML = s;
9.     } else {
10.             document.
   querySelector(“#status”).innerHTML =
    “&lt;h2>No match&lt;/h2>”;
11.         }
12.     }
```

接下来看下另外一个基于索引的Range的用法，通过Range，可以获得某个数据范围之间的数据，比如字母a到字母c之间的所有name，同时还可以设置是否包含或者不包含边界（如是否包含字母a），并且可以对范围之间的数据进行排序，来看例子：

```
1. //大于39
2. var oldRange = IDBKeyRange.lowerBound(39);
3.
4. //大于40
5. var oldRange2 = IDBKeyRange.
   lowerBound(40,true);
6.
7. //小于39
8. var youngRange = IDBKeyRange.
   upperBound(40);
9.
10.     //小于39
11.     var youngRange2 = IDBKeyRange.
   upperBound(39,true);
12.
13.     //20到40之间
14.     var okRange = IDBKeyRange.
   bound(20,40)
```

这里使用的是一个顶层的对象名为IDBKeyRange。其中lowerBound方法指定返回的数据是大于指定的参数的，upperBound则相反，bound则返回指定范围之间的数据。下面给出一个检索的例子，其中在页面表单中可以输入搜索名称的开始和结束范围，如下：

```
1. Starting with: <input type=”text”
   id=”nameSearch”
   placeholder=”Name”><br/>
2. Ending with: <input type=”text”
   id=”nameSearchEnd”
   placeholder=”Name”><br/>
```

```
3. <button id=”getButton”>Get By Name Range</button>
```

其事件代码为：

```
4. function getPeople(e) {
5.     var name = document.querySelector(“#nameSearch”).value;
6.
7.     var endname = document.querySelector(“#nameSearchEnd”).value;
8.     if(name == “” &amp;&amp; endname == “”) return;
9.
10.         var transaction = db.transaction([“people”],”readonly”);
11.         var store = transaction.objectStore(“people”);
12.         var index = store.index(“name”);
13.         //Make the range depending on what type we are doing
14.         var range;
15.         if(name != “” &amp;&amp; endname != “”) {
16.             range = IDBKeyRange.bound(name, endname);
17.         } else if(name == “”) {
18.             range = IDBKeyRange.upperBound(endname);
19.         } else {
20.             range = IDBKeyRange.lowerBound(name);
21.         }
22.         var s = “”;
23.         index.openCursor(range).onsuccess = function(e) {
24.             var cursor = e.target.result;
25.             if(cursor) {
26.                 s += “&lt;h2>Key “+cursor.key+”&lt;/h2>&lt;p>”;
27.                 for(var field in cursor.value) {
28.                     s+= field+”=”+cursor.value[field]+”&lt;br/>”;
29.                 }
30.                 s+=”&lt;/p>”;
31.                 cursor.continue();
32.             }
33.             document.querySelector(“#status”).innerHTML = s;
34.         }
35.     }
```

在上面的代码中，我们首先判断用户到底在哪个文本框中输入了检索条件，然后通过条件判断组合成了range，然后将range传递给打开游标的方法index.openCursor(range)就可以了，IndexedDB则会自动检索出符合条件的记录，例子的代码在test11.html。

在接下来的第二部分的教程中，将探讨包括更新、删除和数组方面的操作，敬请期待。■

本文代码下载参见：https://github.com/tutsplus/working-with-indexeddb

JavaOne 2013：将REST与JSON相结合以创建API

■ Stormpath公司首席技术官Les Hazlewood在JavaOne 2013大会上表示。为了顺应与会观众们的强烈呼声，Hazlewood于本周二重返JavaOne舞台、向大家介绍了如何利用JAX-RS与Jersey创建优秀的REST + JSON API。

创建一套可通过Web进行访问的应用程序编程接口并不困难，但打造一套既运作良好又稳定可靠的API却没那么容易，Stormpath公司首席技术官Les Hazlewood在JavaOne 2013大会上表示。为了顺应与会观众们的强烈呼声，Hazlewood于本周二重返JavaOne舞台、向大家介绍了如何利用JAX-RS与Jersey创建优秀的REST + JSON API。

“一款出色的具象状态传输（简称REST）API从表面上看应该很简单，即使其后端机制实际相当复杂，”Hazlewood在介绍环节之前的记者采访中指出。如果一款API的专注重点在于收集信息并为各条信息提供各自独立的代表，那么只要削减API中的收集及搜索机制（而非罗列所有已经采用的对象），我们就能获得一款简洁而出色的方案，直观而绝不复杂。

Les Hazlewood

在问答环节中，Hazlewood深入探讨了API最佳实践、REST API的优势与劣势以及JavaScript Object Natation（简称JSON）等话题。

为什么Java开发人员乐于尝试REST API?

Les Hazlewood: REST是一套以现有HTTP为基础建立起来的架构风格。在HTTP规范当中，我们交换数据以及创建、读取、更新与删除数据的方式已经被确定下来。在互不相干的设备中创建即读即删信息时，REST能够负责决定特定情况该如何处理。

这正是REST的主要作用；负责此类操作在互不相干的设备间执行时该如何处理。由于REST以HTTP为基础，因此我们可以在Linux设备、Windows设备与苹果Mac设备之间进行通信。总之，REST无需局限于特定平台或者供应商类型——由于HTTP无处不在，因此REST也能够广泛起效。所有编程语言，包括Python、PHP、Java以及C#，都能够与REST并行协作。

从各个方面来说，REST往往由于自身的迷惑性而被看得过于简单。每个人都自认为了解HTTP，因为这正是Web浏览器的表面方式。技术人员了解HTTP协议、GIT以及POST，因为他们已经以Web形式与之接触多年。而由于REST使用HTTP协议，

开发人员会习惯性地低估其实际复杂性。事实上，如今的REST服务已经远远超过XML（过去一直搭配SOAP，即简单对象访问协议）的范畴。

使用REST会遇上哪些难点？

Hazlewood:这正是我来到这里向大家进行介绍的原因。REST是一种架构风格，但其使用方法还缺乏正式的标准或者规范。要了解这种风格，我们需要为其添加注释，否则不同使用者对于功能的理解与效果会出现一定程度的偏差。这是因为REST并非一套能够在设备之间直接照搬的规范，而且人为因素的介入也会引发歧义。在这样的情况下，要正确或者轻松使用REST将变得充满困难。REST与JSON都很简单，HTTP也很简单，但在将这几种机制结合起以解决特定问题时，大家会发现自己很难找到既定的成熟指导。

您在JavaOne介绍环节中建议将JSON与REST相结合，还有其它什么理由吗？

Hazlewood: REST与JSON提供了更为人性化的数据表示方式；数据看起来不再像XML格式那样拥挤；更便于我们直接用肉眼查看。而这正是JSON被广泛采用的真正原因。

JSON是一种语法规则。简单概括，它的内容就是字符串、数字、空、非空。它允许大家以非常简单的格式通过少量元数据表示复杂的情况。我们能够轻松将其使用在多种不同环境之下。它易于机器解析，也同样适合人们直接阅读。

JavaScript是目前世界上最具影响力的重要编程语言之一。即使是在主要由Java、Python或者C#编写的应用程序中，一旦涉及网页或者直观用户界面，那么出场解围的也必然是JavaScript。JSON与JavaScript完全兼容，任何拥有JavaScript编程经验的技术人员都能轻松掌握JSON。如果大家已经跟JavaScript打过交道，那么将其与API相整合也绝非难事。如果API返回JSON且大家已经利用JavaScript编写代码，那么编程语言本身就会了解如何阐释来自服务器的数据。JSON借助人气极盛的JavaScript处理数据交换（而不仅是写入软件）任务。

何时适合使用JSON，何时不适合？

Hazlewood:很显然，XML在数据复制架构方面的表现更出色。XML文档中包含的信息更丰富，而且其中的数据类型划分也更为高效。XML非常适合处理数据交换任务，但糟糕的易用性往往令开发人员望而却步。JSON内容适合直接阅读，XML在设计方面则更为复杂。JSON的语法简单易懂。说起语言设计，JSON中的核心元素非常有限，全部开发成果都以此为基础。由于其出色的简单特性，我们能够轻松对其加以操控及理解。不过在设备消耗信息方面，JSON的表现远不如XML。XML更适合设备直接处理。

原文链接：http://searchsoa.techtarget.com/news/2240205998/JavaOne-2013-Pairing-REST-and-JSON

统治网络：JavaScript的胜利

■ 在以平台多样性为标志的计算时代之下，我们需要能够运行在任何设备浏览器当中的应用程序。原生应用的运行速度也许更快、对于特定平台的功能利用效果也非JavaScript可及，但云时代的来临显著缩小了二者之间的差距。现在摆在开发人员面前的道路有两条：针对桌面系统或者移动平台开发只能运行在单一环境下的应用，或者编写能为任何用户所使用的JavaScript应用。

JavaScript正凭借新型工具与功能提升以极度夸张的速度吞噬整个世界。我们是否应该接受这一无法逆转的趋势?

还记得那些旧日往事吗？很多用户因为担心安全问题而在浏览器中禁用JavaScript。如今这样的担忧已经显得相当可笑——JavaScript已经成为统治网络世界的绝对王者。

在以平台多样性为标志的计算时代之下，我们需要能够运行在任何设备浏览器当中的应用程序。原生应用的运行速度也许更快、对于特定平台的功能利用效果也非JavaScript可及，但云时代的来临显著缩小了二者之间的差距。现在摆在开发人员面前的道路有两条：针对桌面系统或者移动平台开发只能运行在单一环境下的应用，或者编写能为任何用户所使用的JavaScript应用。

当然，以上问题的具体答案取决于应用程序的具体特性。JavaScript拥有明显的局限性——举例来说，出于安全原因，JavaScript无法读取或者写入客户端中的文件。而且“真正”的程序员更倾向于利用自身技术水平在应用中使用大量快捷的开发方式。不过JQuery及其它一系列框架的出现让JavaScript具备了开发高复杂性应用程序的能力，而JSON（即JavaScript 对象表示法）能够实现除XML之外的全部数据传输需求。更不用提Node.js为JavaScript带来的强大服务器端事务处理能力。

JavaScript还能走多远?

巧合的是，InfoWorld网站新技术论坛上的两篇最新博文都在讨论JavaScript的未来发展潜力：一篇由VisiCalc联合创始人兼技术老鸟Dan Bricklin所撰写，另一篇则由Adobe公司的Divya Manian与Thibault Imbert撰写。

在题为《JavaScript在移动平台击败原生代码》的文章中，Bricklin针对JavaScript运行速度低于原生代码的假设提出质疑：

尽管总体来说，原生代码在数学计算方面的执行速度确实要远远胜过JavaScript，但这种观点忽略了众多应用程序之间彼此独立的运行状态。在很多运行过程中，JavaScript应用程序的性能表现往往能够赞同甚至优于原生代码。

为什么会这样？根据Bricklin的解释，这是因为众多杰出的程序员已经花了数年时间对浏览器进行调整，从而实现了运行过程的极端优化。另外，浏览器中已经逐步出现多种先进的运行机制进展，例如3D渲染。

作为Adobe公司的代言者，Manian与Imbert分析了JavaScript的未来前景并提到Mozilla公司的研究项目asm.js。该项目“定义了一套

JavaScript子集，通过编译器生成并利用JavaScript虚拟机实现高度优化。”更令人兴奋的是，他们还探讨了RiverTrail这款由英特尔负责开发的并行编程模型及API——当然也是专为JavaScript所打造。两位作者同时谈到，Adobe公司建议为HTML 5功能引入新的标准化机制，其中包括Regions、Blend Modes以及Shapes等。

通过或明或暗的各种迹象，我们几乎可以断定，Manian与Imbert通过自己的博文给Flash及ActionScript的坟墓又添了一把土。

JavaScript生态系统

JavaScript的胜利宣言绝非凭空想象——几乎每周都会出现新的JavaScript框架以及与编码相关的其它工具。除了jQuery与Node.js，今年InfoWorld网站评选的最佳开源软件奖中出现了七位JavaScript赢家的身影，它们分别是：

◆ AngularJS，一套用于将静态HTML页面转化为JavaScript应用程序的工具集，同时提供对MVC架构的支持。

◆ Backbone.js，一套JavaScript库，帮助开发人员将结构作为模型添加到应用程序及表示数据当中。

◆ Bootstrap，一款响应式Web设计框架，旨在与jQuery相结合。

◆ Enyo，一款面向对象的JavaScript框架，能够被用于创建HTMl 5/CSS应用。

◆ D3，一套JavaScript库，能够在无需涉及插件的前提下在浏览器内实现矢量图形处理。

◆ Ember.js，一款极具发展前景的JavaScript框架，用于开发具备丰富功能的MVC应用程序。

◆ Emscripten，一款将C++代码转化为asm.js形式的编译器，属于由Mozilla公司推出的经过高度优化的JavaScript子集。

上述阵容可谓极度强大——而且这还只是JavaScript麾下力量的冰山一角。这些工具仍处于迅猛的发展态势之中，且大部分属于开源项目，它们的存在将继续推动JavaScript的未来改进。

但是……JavaScript能行吗？？

然而，经验丰富的开发人员对JavaScript仍然有些反感。InfoWorld网站的Andy Oliver就这样阐明了自己的态度：

让JavaScript实现全面普及并非不可能——我们只需要帮自己判断，这到底是不是个好主意……我的顶头上司就很喜欢这种方式，组织起一大群能够利用jQuery以及Node.js的开发人员——他们最好还能简单用用MongoDB等轻量级数据库。但我身边的开发人员显然很抵触这种成天跟JavaScript打交道的工作习惯……项目管理者也不希望把自己的宝贵数据库交到一帮JavaScript开发者的手中。

换句话来说，JavaScript是一种既简单易学、又有些拙手笨脚的编程机制——这将导致编写代码的开发人员陷入混乱、甚至搞不清自己到底在做些什么。

不过杰出JavaScript程序员的队伍正在不断扩张，其中不少人都能在asm.js或者英特尔RiverTrail并行编程模式的辅助下实现诸多先进功能。值得一提的是，他们绝对不会受到工具缺乏这类难题的困扰。

当然，市场上也存在着JavaScript的替代方案，其中最具代表性的就是谷歌Dart——据说该语言将在不久的将来推出1.0正式版本。不过Dart代码需要经过编译才能转化为JavaScript，从而运行在大部分浏览器环境当中。惟一的例外就是谷歌推出的Chromium，它提供Dart虚拟机、因而能够直接与这种新语言相对接。

最好的并不一定总能取得胜利。与x86指令集类似，有时候赢家往往是能够坚持到最后的方案。我不知道JavaScript如何通过自身扩展来迎合开发人员创建应用程序的需求（例如像微软Office这样积淀丰厚的应用），但奇怪的是，通过浏览器窗口实现全平台运行似乎成了未来的必然趋势。我个人可不想刻意跟这种趋势进行对抗。■

英文原文：http://www.infoworld.com/t/application-development/the-triumph-of-javascript-227283

景环境对错误进行过滤，其中包括错误出现的具体行数以及与源代码中的哪一行直接关联。

Aardvark——这款火狐扩展催生了Firebug中的最佳功能之一：允许开发人员查看底层源代码，并且通过将鼠标悬念在HTML页面中的特定元素上来执行各种操作。

MochiKit JavaScript Interpreter——Firebug自己的JavaScript解释器在设计灵感上正是源于这套轻量级JavaScript库，它允许开发人员在访问DOM的同时运行JavaScript命令。

Ratcliffe还在文章中回顾了Firebug扩展的成长历程与发展编年史，对于在编程工作中经常使用这款工具的朋友来说，这绝对是一篇不容错过的精彩论述。■

链接

少了这些工具，JavaScript将变得更难于使用

作为一款人气极高的Web开发调试工具，Firebug（萤火虫）的设计灵感源自之前出现的多款实用程序。

现实世界中的萤火虫对Web开发工作可没什么帮助。

作为一位Web开发人员，大家几乎肯定听说过Firebug的名头。但对它的历史，各位也许就不甚了然了。如果感兴趣，朋友们不妨点击此处阅读Mozilla公司开发人员Mike Ratcliffe最近撰写的回顾性文章，其中详细描述了Firebug的前世今生。

Firebug于2006年由Joe Hewitt一手打造，他同时也是火狐浏览器的原始开发者之一。根据Ratcliffe的说明，以下几款扩展及工具给Firebug的不同组件带来或直接或间接的重大启发。

Venkman JavaScript Debugger——这款JavaScript调试工具专门针对Mozilla出品的浏览器，创建于2001年（没错，其名称正是来自Bill Murray在<捉鬼敢死队>中塑造的角色‘Peter Venkman博士’），并成为Firebug中JavaScript调试工具的原形。

View Source Chart——Firebug的HTML层正是基于这款火狐插件，作用是以分层方式显示底层HTML。另外，分组标签也被嵌入到类似的层当中。

Console ——很多人更乐于将其直接称为Console平方，它直接启发Firebug创造出自己的Console层。它能够通过类型、语言及背

十大不容错过的热门JavaScript框架项目

□ 核子可乐/译

技术社区的力量在用户选择JavaScript框架时发挥着至关重要的作用。在今天的文章中，我们将一同了解支撑AngularJS、Backbone.JS以及Ember.JS等项目的背后推手。

想要弄清楚到底哪种JavaScript UI框架能够切实满足项目及组织的实际需求？面对十大人气方案，我们还有很多需要认真考虑的因素。

在过去几年里，我们亲眼目睹了UX库及框架的迅猛发展——其中大部分属于开源技术成果。目前市场上存在大量用于比较各类框架完整性的方法，但人们往往忽视了几大至关重要的决定性因素——这些开源项目背后的技术社区与生态系统。技术社区与生态系统的规模、实力与发展走势将最终决定开源项目的未来命运。总而言之，这些统计结果应该成为指导决策的关键性依据。

今年早些时候，我曾经有幸参与了一次调查，其主题为“2012年以来发展势头最迅猛的开源项目”。最终的排行结果令我震惊，榜单上几乎挤满了各类JavaScript项目。这激发了我的好奇心，因此我决定深入了解特定JavaScript库/框架项目，看看这些专注于帮助开发人员创建出丰富且高扩展性用户界面的项目到底为何拥有如此广阔的发展空间。我在分析中使用的开源社区元数据来自Ohloh.net以及GitHub，其中包括各项目的星级评分（用于帮助用户及时掌握感兴趣的资源库的最新动态）以及个别项目所吸引到的拥护者人数。

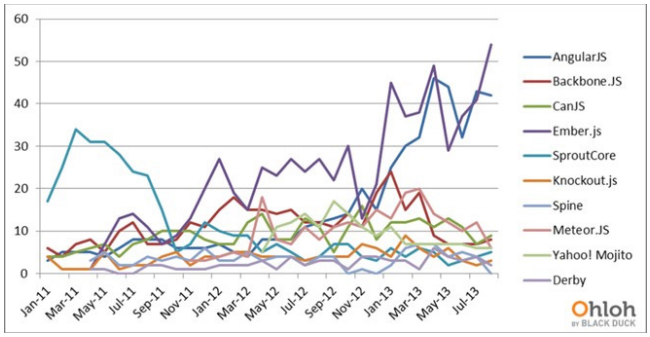
哪些项目榜上有名

在本次调查分析中，我关注的主要是十大开源项目背后的技术社区。顺带一提，这十大开源项目涵盖了UI库、框架以及包括服务器端运行时间在内的全堆栈框架。此次分析的对象全部为当下的热门UI项目，例如AngularJS、Ember.JS、SproutCore、Backbone.JS、Knockout.js、Spine、CanJS、Meteor-JS、Derby以及Yahoo Mojito。

我的目标是描绘出一幅关于各项目相关技术社区的发展图谱，尤其是在规模与发展速度方面，并将此与个别项目的普及范围及成功程度加以比照。当然，我还在密切关注各个项目所对应的生态系统，希望找到生态系统状况与项目普及范围及成功程度之间的联系。

每月贡献者数量

探寻各个项目每月贡献者的具体数量及增长幅度有助于了解该项目的发展态势，并足以了解业务对该项目的关注程度外加关键性发展临界点。下面的图表中囊括了十大项目的每月贡献者参与情况，数字代表的是各个月份提交代码的贡献者数量。



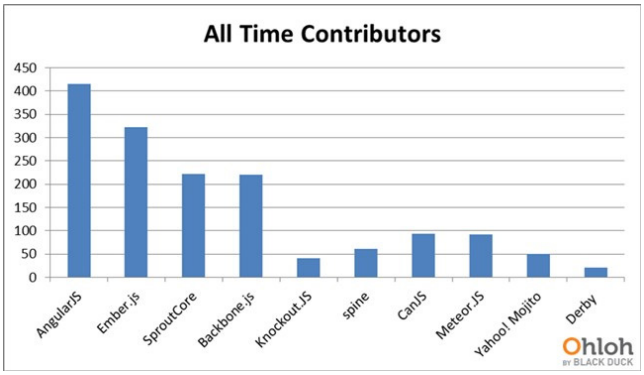
到，贡献者爆炸式增长的历史转折点出现在2011年12月，当时该项目刚刚脱离SproutCore并独

立出来。谷歌的AngularJS的贡献者数量同样走势喜人。

如大家所见，SproutCore曾经在发展初期领跑每月贡献者数量榜。然而随着2011年12月SproutCore的分裂与Ember的诞生，技术团队中的大部分开发人员都选择了Ember作为自己的新起点。如今Ember与AngularJS一道成为2013年中贡献者数量最多的项目，且二者在全局技术社区的发展态势方面同样表现优异。需要注意的是，今年春季Backbone与Meteor的每月贡献者数量出现显著下滑，而此时正是AngularJS与Ember极速上升的时段。

历史贡献者数量

下面我们来看项目整个发展历程中的总体贡献者数量，这能帮助我们从另一个视角找到合理的结论。值得强调的是，历史贡献者数量往往反映出项目的具体管理风格。很多项目由小型团队进行严格管理，其它项目则采取完全开放的政策、接受技术业界所带来的多样化元素。

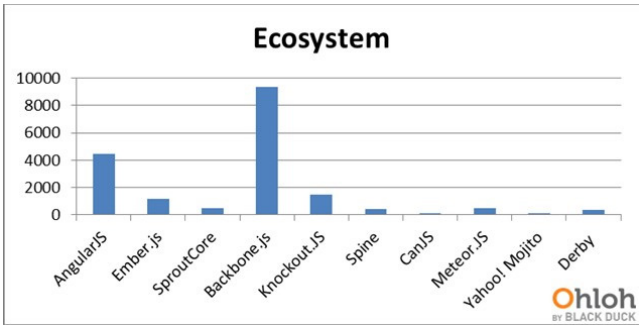


项目发展历程中的总体贡献者数量：AngularJS与Ember拥有最庞大的全局技术社区，这充分反映了二者在近一段时间以来出色的每月贡献状况。

历史贡献者数量还能帮助我们理解项目在任何特定时段的相对发展趋势。规模庞大、制度完善的项目往往拥有可观的贡献者基础，他们帮助修复漏洞、制作说明文档并完成大量其它任务。贡献者总数往往与项目代码基础的规模密切相关。与其它技术社区指标相比，例如代码行数、总提交数量以及每月提交数量，历史贡献者数量更能说明问题。

项目生态系统

除了当前贡献者数量之外，项目技术社区的力量还体现在生态系统方面。生态系统的建立与扩展同项目本身存在着千丝万缕的联系。这意味着单靠核心技术社区，我们还无法一窥项目的真实全貌。只有将视角放在更加广阔的生态系统身上，大家才能够对单一项目的全面成效做出准确判断。



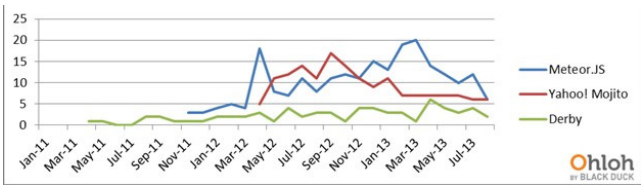
相关项目数量：开源项目的稳定性可以通过与之相关的周边项目数量来衡量。在这方面，Backbone.js与AngularJS是当之无愧的赢家。

值得注意的是，Backbone拥有庞大的生态系统，这从侧面显示了其在技术业界的普及程度及发展态势。为了进一步验证Backbone生态系统的发达程度，我在GitHub上过滤掉所有评价不足三颗星的内容、只保留三星及心目且与Backbone密切相关的项目——仍然找到了1627个结果。相比之下，AngularJS的三星及心目相

关项目只有个，这意味着Backbone的生态系统规模仍然达到AngularJS的两倍以上。

全堆栈解决方案

由于全堆栈解决方案的审视角度与其它项目有所差别，因此我希望将这类项目单独划分并进行比较。全堆栈解决方案中既包含客户端框架也包含服务器端框架；由此可以假设，此类方案的相关代码数量更多、随着时间推移参与进来的技术人员规模也更为庞大。下面这份图表显示的正是这些项目在相对较早的发展时期内的参与情况。



全堆栈解决方案数量：Meteor与Mojito对于技术社区的吸引力似乎更强，至少从贡献者数量的角度来看是这样。

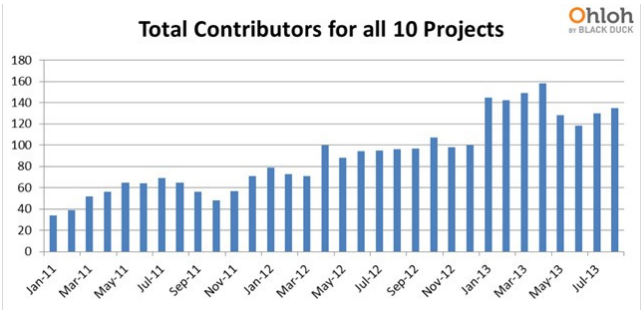
JavaScript的崛起

将时间与精力用于创建结构化JavaScript方案的开发人员数量正持续激增。以下图表显示的正是自2011年年初至今，参与代码贡献的开发人员数量。统计结果反映出如今UI框架在应用程序开发工作中的重要地位，同时也说明不同规模的企业对于差异化用例的方案需求正日益强烈。自今年一月份开始，参与代码贡献的开发者数量迎来显著增长，这很可能是受到了AngularJS与Ember团队的带动。

十大JavaScript项目的代码贡献者总数：如果大家抱有任何疑问，这份图表将用事实证明JavaScript正以改天换地之势席卷全球。另外，与2011年相比，今年各项目的平均贡献参与者数量普遍翻倍。

代码行与提交数量

大家可能希望进一步了解各大开源项目的代码基础规模——以及代码基础与代码贡献者数量之间的关联。通过比较，我们发现技术社区规模与总体提交数量几乎跟代码基础规模没啥关系。举例来说，AngularJS项目中每位贡献者平均带来413行代码，而Ember项目中每位贡献者平均只带来146行代码。



码行数与贡献者数量：每位贡献者提供的代码行数越多，就说明该项目的技术难度越低——某些人将此视为项目健康程度的一项考量指标。

不过大家也可以在技术社区规模（即历史贡献者总数）与历史提交数量之间找到关联，并以此为基础分析每位贡献者为项目带来的平均提交数量。统计结果可能会告诉我们哪些项目更易于做出贡献，这又会给项目的长期发展带来哪些影响。

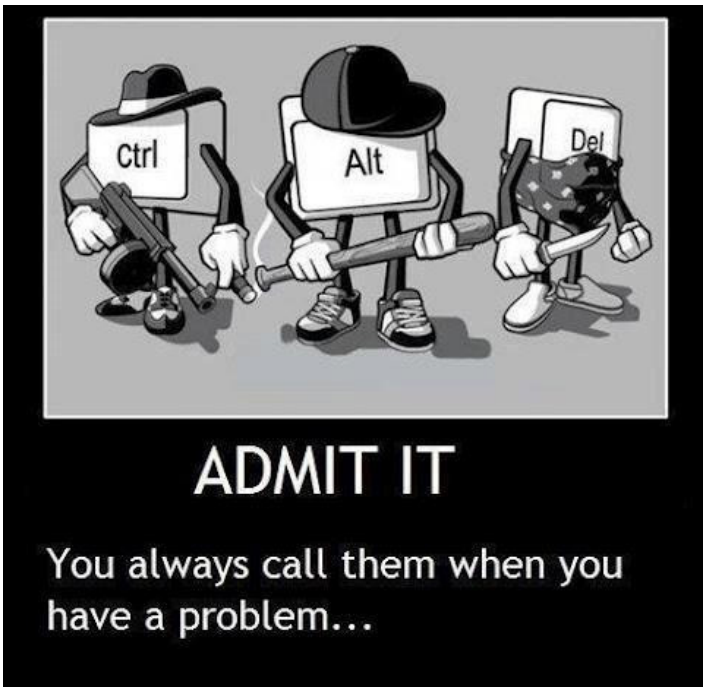
预见未来

技术社区规模与发展速度对于项目的前景而言非常重要。尽管这些指标无法向我们昭示整条轨迹的来龙去脉，但我们仍然可以借此了解技术人员对特定社区的参与程度、从而帮助自身了解项目的可行性及可持续性。它们同时也能帮助大家找到项目生命周期中的各类增长拐点，从而找到

参与项目并对其产生影响的最佳时机。

UI框架是我最熟悉也最喜爱的主题，这是因为我曾在Adobe公司用五年时间打理Flex框架的产品营销活动——现在该框架已经转型为Apache项目。基于前面提到的各项数据分析，我的结论是Ember与AngularJS是近来崛起的框架当中最值得关注的竞争选手。不过在另一方面，Backbone那庞大而活跃的技术社区则昭示了该项目作为UI方案的极高人气，相信它将拥有可持续且光明的未来。■

原文链接：http://www.infoworld.com/d/application-development/the-10-hottest-javascript-framework-projects-228335



趣 文

盖茨承认 Ctrl+Alt+Del 是失误

这样的按键组合原本是为了防止其他程序伪装登陆提示框，从而窃取用户密码。

据国外媒体报道，微软创始人比尔·盖茨（Bill Gates）最终承认，强迫用户按Ctrl+Alt+Del三个组合键登陆电脑的做法是个错误。

在哈佛大学筹款活动中接受采访时，盖茨提及当年创建微软时的一些事情，其中包括设置Ctrl+Alt+Del组合键的初衷。盖茨说，这样的按键组合是为了防止其他程序伪装登陆提示框，从而窃取用户密码。

“这是一个错误，”盖茨向听众解释，“我们可以利用单一的按键，但IBM键盘设计人员没有给我们设计这样一个按钮。”

这种古怪启动电脑的按键组合由IBM PC的工程师大卫·布拉德利(David Bradley)设计。布拉德利在之前的采访中称：“没错，是我设计的，但盖茨让它家喻户晓。”这番话让盖茨显得相当尴尬。

时至今日，这一组合按键方式也被Windows 8采用。尽管Windows 8有了新的登陆界面，但是传统的Ctrl+Alt+Del按键组合方式依然管用。

据悉，Ctrl+Alt+Del不是微软承认的唯一失误。本年初，微软表示，他们没有抓住机会进入移动市场。盖茨说：“我们错过了手机，这让我们失去领导地位。”鲍尔默最近同样表示：“我们过去一直纠结在Windows系统上，没有抽出精力去研究手机。”■

四种有能力取代Cookies的客户端Web存储方案

□ 廖煜嵘/译

目前在用户的网络浏览器中保存大量数据需要遵循几大现有标准，每一种标准都拥有自己的优势、短板、独特的W3C标准化状态以及浏览器支持级别。但无论如何，这些标准的实际表现都优于广泛存在的cookies机制。

今天的Web应用程序开始在客户端中执行大量数据处理工作，甚至可能需要以脱机方式完成任务。可以说，客户端数据存储对于下一代Web应用程序的发展起到了至关重要的作用。

然而直到现在，cookies仍然是用户浏览器中最常见的数据存储机制。如果一款Web应用需要重复访问某些数据，则只有两种方式可供选择：要么再次向服务器发送请求以获取数据，要么读取保存在cookies中的内容。

cookies机制只能提供有限的存储空间——最多4K或者4096字节——因此总量较大的数据会被拆分成4K大小的块从而加以明确而直接地管理。

但这种方式对于存储的协作及管理而言显然并不可行，因此我们需要拿出一套新的替代性方案。

cookies的承受能力太过孱弱

网络浏览器最初只是通过HTTP并解析HTML实现应用程序对文档内容的加载。但在此后不久，第一款网景浏览器出现了，它满足了用户的一系列实际需求，但却需要利用本质上无状态的HTTP协议来通过某些机制实现状态追踪。面对这一问题，Lou Montulli于1994年创造了浏览器cookie（当初被称为‘magic cookie’），并首次亮相于Mosiac网景浏览器0.9b版本之上。

在通用网关接口（简称CGI）提供的服务器端脚本访问功能与cookies的共同辅助下，最早的Web应用程序终于变成现实。最终，我们开始沿着这条小路将浏览器转化成为一种通用的应用程序平台。

然而cookies机制存在着严重缺陷。正如前面所提到，它只能存储极少量数据，而且很容易受到各类攻击活动的影响，这样的状况让我们很难利用其存储个人信息及敏感数据、从而极大限制了它的使用范围。

cookies会介入到从浏览器发向服务器端的每一条HTTP请求当中。假设一个网页中包含的四张图片、一个外部CSS文档外加JavaScript文档。系统会为该域设置一个4K的cookie，浏览器则分四次将该cookie转发至服务器端——一次针对HTML页面、一次针对每张图片、一次针对CSS文档再加上一次针对JavaScript文档。

令问题进一步复杂化的原因在于，这个理论上为4K大小的cookie需要从浏览器端传输至服务器

端；由于大部分用户使用的是异步互联网连接，即上传速度低于下载速度，因此在HTTP响应头中传输cookie数据一定会造成不必要的带宽占用。

由于上述限制因素的存在，大部分cookies的体积都要远小于4K。谷歌建议每个cookie的实际大小不要超过400字节（或者200个字符），从而实现最佳性能表现。他们还建议称，在图片、CSS以及JavaScript等来自独特域的静态文件应该禁用cookies机制。

由于cookies机制在本地存储领域存在诸多问题，目前已经出现一系列新兴方案，旨在拨乱反正、保质保量完成任务。近几个月以来，已经有两款方案走上正轨、得到W3C的强烈推荐——它们能够很好、甚至比我们预想中更好地帮助浏览器支持本地存储功能。

目前我们可以从四种主流客户端数据存储机制中做出选择，它们分别是：Web SQL、IndexedDB、Web Storage以及Application Cache。下面我们就逐一对每套方案加以评述，并探讨它们在运作及效果方面的各自特性。

Web SQL: 擅长（但是否有些过时？）数据库创建与执行

Web SQL是一种利用数据库进行数据存储并利用SQL处理检索任务的API。最近，Safari、Chrome以及Opera等知名浏览器纷纷在Web SQL与IndexedDB的竞争之中选择了前者。不过2010年时，SQLite还是惟一一款能够与Web SQL协作的数据库，而W3C出于安装基础较小的理由而停止对这套方案进行支持。

Web SQL的工作机制相当新奇，下面我们就一起来看示例代码。

Web SQL数据库的使用感受与关系类数据库及SQL非常相似。使用这款数据库的第一步在于创建并打开。如果大家不希望额外创建一套数据库，那么完全可以直接开始使用，API本身会自动完成创建工作。

下面我们来看一部分用于数据库创建的代码：

```
1. var db = openDatabase( ‘cats’ , ‘1.0’ , ‘a
   catalog of my cats’ , 2 *1024 * 1024);
```

按照从左到右的顺序，openDatabase后面的参数依次代表着数据库名称、版本号、文字说明以及预计数据库大小。

数据库创建完成之后，大家就可以着手使用了。在WebSQL数据库上执行SQL与创建事务对象并加以执行一样简单：

```
1. db.transaction(function (tx) {
2.  tx.executeSql( ‘CREATE TABLE cats (id
    unique, name)’ );
3.  tx.executeSql( ‘INSERT INTO cats (id,name)
    VALUES (1,” Mr. Jones” )’ );
4.  });
```

尽管Safari、Chrome、Opera以及Mobile Safari都支持这款API，但自2010年以来Web SQL就没有发生过任何变化，因此它不太可能成为本地存储的新型标准。

Web Storage：取cookies所长、去cookies所短

Web Storage利用一种简单的方法在用户的浏览器中存储键/值对。但它与cookies之间的相似之处也就仅此而已了。

◆ Web Storage是一套持久性方案。一旦某个值被存储之后就不会再消失或者终止，除非被应用程序或用户明确删除。

◆ Web Storage能够处理大量数据。目前浏览器的总体存储区域大小最高为5MB。

◆ Web Storage无需依赖于服务器，而且不必向服务器端发送数据。当然，大家可以随意实现本地化数据存储并将其与服务器进行异步式同步，但Web Storage的表现始终出色而且在离线与在线状况下都能正常生效。

◆ Web Storage提供四种主要方法——getItem（键）；setItem（键、值）；removeItem（键）以及clear（）。

最后，Web Storage包含两种完全不同的存储类型：SessionStorage以及LocalStorage。

SessionStorage的作用在于保证被保存在当前浏览器窗口当中的数据仅作用于该窗口。举例来说，当大家使用电子商务类应用程序时，利用SessionStorage来记录用户的购物车信息能够避免误操作所带来的二次购买状况。

下面再来看LocalStorage，它专门负责保存可同时作用于同一浏览器之下各窗口及标签之间的数据。因此，如果大家在Chrome当中打开了三个关于同一网站的窗口，那么三者能够共同使用同一套LocalStorage容器。相比之下，如果我们打开三个内容彼此独立的网站窗口，那么每一个都将使用彼此独立的容器。同样，如果大家在不同的浏览器当中打开同一个网站，那么每种浏览器都需要使用属于自己的容器，因此无法共享同一套通用的运行环境。

要设置一套新的键-值对并进行检索，大家可以使用下列JavaScript命令：

```
1. //first set firstname equal to Sparky.
2. localStorage.setItem( “firstname”,
   “Sparky” );
3.
4. //next, get the value of firstname (hint, it will
   be Sparky).
5. localStorage.getItem( “firstname” );
```

今年夏天，Web Storage API正式获得W3C推荐标准这一殊荣。展望未来，Web Storage完全有可能在一切原本cookies发挥作用的舞台上成为新的处理方案。

但Web Storage能做的还很多。如果大家的数据集并不太大，Web Storage还提供另一种可能是最为简便的处理办法——甚至比cookies更简便——从而顺利搞定浏览器中键-值对的设置与检索工作。

IndexedDB：可搜索且不存在文件大小限制

Indexed Database是一款利用索引化事务性数据库对用户计算机上的数据进行保存与索引的API。IndexedDB带来更快速、更精妙的数据存储与检索效果，在这方面采用简单键-值对存储机制的cookies以及Web Storage都只能甘拜下风。

与Web Storage一样，IndexedDB API在今年夏天（也就是2013年7月）向Web标准迈进了一大步，成为W3C候选推荐名单中的一员。

与Web Storage相比，IndexedDB带来四项具体提升：

◆ 能够对索引数据进行高效搜索。

◆ 数据库能够将多个值保存为一个键，而键-值机制则要求每个键都必须惟一。

◆ 事务型数据库提供多项针对系统及应用程序故障的保护措施。如果事务流程未能正常完成，则将通过回滚方式进行恢复。

◆ IndexedDB数据库对数据内容的大小不加限制。在火狐当中，浏览器会要求利用权限将数据库的容量提升到超过50MB，而IndexedDB的实际数据存储量限制直接取决于分卷或者磁盘驱动器本身的容量极限。

除了Safari之外的所有主流浏览器都已经支持IndexedDB。不过由于Safari支持Web SQL，因此我们完全可以利用IndexedDB夹层（或者被称为shim）通过Web SQL实现IndexedDB的功能与语法。

要使用IndexedDB，第一步需要打开一套数据库。

```
6. var request = indexedDB.
   open(“myDatabase”);
```

在数据库创建完成之后，大家可以创建一个存储对象（与表格非常类似）并向其中添加数据。假设我们需要向其中添加如下数据：

```
1. const petData = [
2. { id: “00-01”, firstname: “Butters”, age: 2,
   type: “dog” },
3.
4. { id: “00-02”, firstname: “Sammy”, age: 2,
   type: “dog” }
5. ];
```

接下来，我们可以创建数据存储机制并通过下列代码加以使用。请注意onupgradeneeded的处理方式：我们在改变数据库结构时需要用到这一方法。

```
1. request.onupgradeneeded = function(event) {
2.  var db = event.target.result;
3.  var objectStore = db.
    createObjectStore(“customers”, {keyPath:
     “id”});
4.  for (var i in customerData) {
5.  objectStore.add(customerData[i]);
6.  }
7.  }
```

IndexedDB擅长于搜索大型数据库集，并能够通过将结构化数据移动至客户端来提高Web应用程序的性能表现。目前它已经非常接近W3C的推荐级别，而且能够被用于全部浏览器平台——尽管具体实施方式有所区别，如前文所述，在Safari中需要借用夹层机制。

Application Cache: 让离线客户端存储成为现实

Application Cache与前面提到的其它客户端数据存储APi都不一样，但它同样值得关注，因为它已经成为离线客户端Web应用程序的重要组成部分。

Application Cache使用的是一套缓存列表。所谓列表，只是一个非常简单的文本文档，其中列举了所有应该或不应该通过缓存机制处理的资源条目，从而指导浏览器下载特定文件、加以保存并在必要时予以使用——而不必再向服务器发出重复请求。目前所有主流网络浏览器都支持Application Cache机制。

要使用Application Cache，我们需要首先在包含有缓存对象文件的网站中保存一个扩展名为为.appcache的文本文件。根据所使用Web服务器的具体类型，我们可能需要为.appcache文件创建一个自定义MIME类型以确保它们能够正确作用于浏览器并可被作为应用程序缓存文件读取。

下面我们列举一个缓存列表文件作为范例：

```
1. CACHE MANIFEST
2.
3. CACHE:
4. /css/styles.css
5. /js/javascript.css
6. /img/logo.gif
7.
8. FALLBACK:
9. /img/weathertoday.png /img/
   weathernotavailable.png
10.
11.      NETWORK:
12.      *
```

现在我们来详细解读其中的内容：

◆ CACHE部分用于告知浏览器哪些资源需要进入缓存以实现离线查看。这些文件会一直保留于缓存当中中，直到缓存列表发生变化。请记住这项要求，非常重要。

◆ FALLBACK部分则用于告知浏览器哪些要显示的文件会取代非缓存资源。举例来说，在上面的FALLBACK部分中，我们可以推测如果latestweather.png图片无法被正确下载，那么当前天气状况无法在离线状态下实现图片显示。

◆ NETWORK部分用于告知浏览器哪些资源只能通过在线模式进行获取。结尾部分的星号表示目前缓存中不存在任何一种网络资源。

Application Cache是一款出色的工具，只要使用得当、它几乎没有什么缺点。其实正确使用是一门学问：如果大家单纯把网站上的所有内容都添加到缓存当中，那么访问者们会很快发现网站内容永远不会发生变化。如果大家只把变化频率不高的内容保存在缓存当中，或者努力保证缓存列表始终处于最新并在上传文件后及时发布新的列表版本，那么Application Cache将带来几乎与在线模式无异的出色离线应用程序运行效果。

本地浏览器存储在过去几年中迎来了一轮重大变革。不同API及推荐项目所使用的多种多样而且彼此相近的名称让我们很难弄清哪些可以继续使用、而哪些应该及时淘汰。总而言之，浏览器数据存储领域拥有多种不同方式可供选择，而且每一种都有非常充分的存在价值。

无论如何，开发人员们努力通过cookies向服务器发送简单的小型名-值对的时代已经结束。今天，我们拥有更多优秀的方案可供使用。■

原文链接：http://www.cio.com/article/739064/4_Client_Side_Web_Storage_Options_That_Replace_Cookies

专访本土数据库CTO武新：谈如何发力细分大数据市场

“棱镜门”的爆发,让爱德华·斯诺登从默默无闻成为头版头条，也为蓬勃发展的大数据产业笼罩上了一层奇特的光晕。在这场众说纷纭罗生门下，大数据话题被推到了风口浪尖，国内的大数据市场也因此变得愈加火热。51CTO记者特别采访到南大通用数据技术有限公司的首席技术官武新，和我们分享了大数据的过去、现在、将来和南大通用目前在数据分析、数据挖掘等方向的开发现状和未来目标。

武新,南大通用公司高级副总裁兼CTO，法国奥尔良大学博士。国家“千人计划”专家。毕业于法国奥尔良大学，有20年的从业经验，在著名的甲骨文（ORACLE）任职12年，是最早获得甲骨文 公司Oracle Certified DBA的数据库管理专家。武新于2010年获得中组部实施的国家“千人计划”荣誉，是工信部认可的数据库专家。2008年7月，武新回到国内，任南大通用高级副总裁兼CTO，是南大通用GBase 8a 分析型数据库及其配套工具总设计师。

“大数据”概念价值提升，带热数据产业链

数据本身是什么，我们并不陌生。IT经济社会出现之后，数据成了大家火热关注的问题。从行业角度看，在互联网高速发展的十几年中，数据处理技术日新月异，加上移动互联和物联网技术和商业模式的新机遇，加速了数据的产生速度，数据存储量开始爆炸式增长。“大数据”概念应运而生。

然而“大数据”概念出现之前，数据分析、数据处理等数据库领域技术在不温不火中持续发展。也出现了数据仓库、BI等新技术概念。但从媒体角度看却没有获得关注焦点。直到“大数据”概念出现，将整个数据领域推至最高点，成为全球关注的热点概念。

对于这一现象，武新表示：互联网的出现，从技术角度和商业模式上颠覆了传统行业的经营状况，我们每个人的生活方式，也在互联网和移动互联网的推动下发生根本变化。除去概念炒作的影响外，可以说“大数据”概念提升了数据本身的价值。数据本身没有模型，或者说没有一个量化的方法。因此，我们无法给出一个明确的价

格；但是，在大数据的推动下，企业对数据的重视程度进一步提升，让我们看到了数据的价值体现和资源地位。

除此之外，数据仓库、BI等早早出现的技术，在“大数据”的带动下在应用上更加活跃。接下来的大数据时代，是人类信息社会的收官阶段。之前的计算机时代和互联网时代，都是为大数据时代做铺垫和准备的。计算机时代的核心是计算能力，极大提高了人们对数据的处理能力；互联网时代解决了信息移动和连接的问题；而大数据时代，可将世界万事万物通通数据化，让人们在数据利用中优化现实操作和行为，令全球系统的运行更为高效。

所以说“大数据”的出现，不仅开启了数据领域的极速发展。对该领域的开发者而言，也迎来了最佳发展阶段。

多方面因素，促使数据分析使用门槛降低

行业里面有这样一种说法：“大数据分析是有钱人的游戏”。

关于这一说法，武新谈了自己的观点：“如果时间倒退5年，这个观点是成立的。在过去，我们去做数据仓库，做BI，确实需要很大的投资，不仅是在软件和硬件的大量投入；在高端人才的招揽上，也要投入大量的资金和精力。但是，随着互联网行业的推动，数据生产速度加快，数据分析和数据处理技术也日益完善，大数据分析的门槛慢慢降低。究其原因，主要有三点。

- 云计算的出现
- 互联网技术的飞速发展，开源力量凸显
- 大批高技能人才涌现

基于以上原因，大数据分析所需投入资源下降，国内各大行业公司普遍使用大数据分析技术。然而，随之而来的问题就是，大数据市场的竞争状态加据，单位生存空间变窄。如何定位自我位置，抓取独特身份，显得尤为重要，也成了数据库公司的思考难题。”

错位竞争，特色产品面对专用市场

“错位竞争”，特色产品面对专用市场，是南大通用的整体战略定位。

武新解释说：“南大通用创立之初，董事长崔维力先生提出了这样的战略方式。我们看到，在传统数据库市场IBM、微软等几家大型公司占据了几乎整个市场份额，在行存储技术领域做到了极致，技术市场达到了饱和状态。因此，在这种情况下，我们很难在传统市场里分得一杯羹。但是在新兴的数据分析领域，我们可以与国际巨头站在同一个起跑线，，我们的产品可以在市场上比他们表现的还要好。这就是所谓的‘错位竞争’，做专用数据库，发力细分市场。”

做为专业数据库产品，在存储方式上，南大通用采用列存储模式。在数据上，更快捷的进行聚合、增组、关联；更加便于进行大规模的数据分析、数据统计。对IO的要求也大大下降，拥有较高的数据压缩比，适合做B型运算。在架构上，不同于传统数据库的垂直架构，而是像Hadoop一样的横向扩展，相对于传统数据来说在计算能力有明显的优势。

在过去20年，几乎一种数据库平台，就能满足所有应用类型。但是，随着数据类型的细分，这样的数据处理模式渐渐无法满足用户需求，产

生越来越多瓶颈。演变到现在，数据处理和应用形成了朝细分市场发展的模式，再次肯定了南大通用战略方针的正确性。所谓细分，就是对某一类数据或某一类应用，做专门的处理技术。精通特定领域的数据分析，特色产品面对专用市场 ，根据不同需求，做不同产品。

专用数据库产品：分析数据库GBase 8a

GBase 8a，是南大通用投入最大的一款分析数据库产品。GBase 8a采用了列存、智能查询、高效压缩、双向并行、自适应优化等多项新技术，打破了以往提高性能只能靠增加数据库的容量，建很多索引的常规，使得GBase 8a既有高性能又有很高的数据压缩比。

武新指出：“经过用户实际测试，在典型分析型应用中表现出：1、高性价比：几乎不用调优就可以达到高性能，不需要考虑如何建索引，如何分区等问题。占有磁盘空间大大降低,节省大量存 储设备费用,是传统数据库的1/5甚至更高，使用通用、中低端的存储设备和服务器就可以达到很高的性能;2、高性能: 与国际传统数据库相比在批量聚集、统计性能;即席查询性能、模糊查询性能等方面都有几倍到几十倍的提高;3、高可用性：安装、调优、维护、扩展非常简单， 好用。”

添加非结构性数据处理技术，帮助用户解决Hadoop平台问题

经过几十年信息化发展，传统行业用户积累了大量数据。其中结构化数据占大多数。ERP等各种系统产生的数据，也基本上都是结构化数据类型。然而最近几年，我们不难发现，半结构化数据和非结构化数据数量迅猛增长，尤其是半结构化数据的数量。

对此，武新认为：“今天的大数据概念里，从数据特征看，半结构化数据和非结构化数据的比重占到90%以上。半结构化数据，有类似文档这样的东西；非结构化数据以视频为主。针对这一类数据的处理，基于Hadoop的平台更为擅长。目前，我们Gbase 8a集群，已经将全文检索—一种半结构化数据处理技术，加入到其中去；在一步一步的朝处理半结构化、非结构化数据的方向发展。我们的目标，要做一个面向企业和行业，全数据处理产品和平台。未来我们的研发目标，将把非结构化数据的处理，做为首要任务。尽可能多的跟开源进行对接。因为在这个领域，开源已经有了很多非常优秀的内容。对接之后，以这两种技术的优势，为用户提供一个平台，全面处理各种类型数据。”

Hadoop 分布式计算平台以其在处理海量数据中的高可靠性、高扩展性等诸多优点，得到了大家的广泛认可和共识。Hadoop作为一个云计算平台，它的出现，解决了单个PC机计算能力薄弱的问题，可以同时在几百、几千个PC机上提供强大的计算能力。在企业中，作为一种ETL工具，在处理海量数据上，有着非常明显的优势。传统的数据库无法做到。在复杂的数据模型挖掘、预测模型的计算上，也占有霸主地位。但是，不可否认的是，在使用过程中，依然存在某些问题。这也是南大通在结合这样一个产品的同时，亟需解决的问题之一。

南大通用的的很多用户已经开始尝试使用Hadoop技术进行数据处理和进行一些项目实验。武新表示：“面对这样一个优秀的平台，我们所要做的有两点：第一，努力将Hadoop平台

企业化。Hadoop企业化，就是要把Hadoop平台变成一款真正的产品，更加方便的供用户使用；未来几年或许能实现，但就目前而言还没有达到这样的成熟度。第二，解决技术更新和用户想要稳定环境之间的矛盾，为用户的稳定使用提供服务。”

坚持通用数据库发展，开始新技术应用实践

目前为止，国内行业大数据市场中，政府的金融、电信等行业的整个IT架构，还是建立在基于小型机的传统架构基础上；核心业务，依旧采用传统数据库模式。随着非结构和半结构数据的大规模增长，这一两年开始，一部分架构开始尝试基于分布式计算模型，例如Hadoop、MPP等技术的尝试。对于这一变化趋势，武新认为：“Hadopp、MPP等分布式计算模型在处理某些业务上的明显优势，促使金融、电信领域愿意去尝试这项技术。除此之外，在数据量上，无论是金融、电信还是政府部门，都在进行从TB到PB的级别迁升；数据量级上将迎来一个新的里程碑，所以对数据处理技术上的要求，也促使了他们投入分布式计算模型的应用的步伐。”

对于未来的国内数据库市场发展，虽然微软、IBM等公司占据了整个市场，但在通用数据库产品上，仍然要坚持去投入。武新觉得：“经过几十年的技术积累，国内数据分析领域在数据仓库、BI等技术上有了绝对性的建设。目前要做的，是思考如何进行行业转型？新一代数据仓库怎么做？新一代BI是什么样子的？在BI方面，南大通用也将进行投入，做一款动态BI产品，实现人机交互模式，这也是未来大数据发展的方向，对传统BI也将起到良好的推进作用。”

因此，国产数据库在继续投入通用数据库市场的同时，努力开发下一代新型技术，在新的数据分析领域开辟自己的领地，在激烈的数据库市场中切割出自己的细分市场，是提升自我市场竞争力和改进自身现状的最佳选择。■

编者按

网络时代的今日，浏览器成为我们每日必不可少的工具。琳琅满目的浏览器市场，飞速的浏览器更新速度，让我们有了更多的选择和更挑剔的体验要求。

DevOps进化论：新时代的土豪

DevOps一出现，很多人就猜的八九不离十。望文生义嘛，这个可以理解，但是如果单纯的从“研发团队”与“运维团队”之间的那点事儿说的话未免太狭隘了。现在大家的关注慢慢从最开始的开发转移到了狭义当中来讲DevOps的概念或者方法论，特别是创业中的企业或者中小型企业纷纷效仿，慢慢的觉得这个概念很好，从自动化代替了大部分开发人力的时候，短期的见效很快，然后就会开始量化。但是当更高的业务端去扩展的时候，慢慢的发现财力（新技术财力，新角色财力，工具购买财力等等）支撑不住了，又开始去抱怨。当然，土豪公司除外。

再看前段时间，IBM收购DevOp(开发与运营合作)解决方案提供商UrbanCode，而UrbanCode狭义的定义DevOps这个领域，它实际上是全球的NO.1，而在去年1月也收购了Greenhat公司，对其在测试领域中的DevOP概念形成了支持；再去年5月收购了Tealeaf公司，则是在反馈领域中加强了这种概念。那么这一连串的收购，IBM真正的目的是为了什么？小编认为，IBM真正的目的就是为了加快DevOps的落地，减少开发流程费用，把开发测试打通到运维发布端，稳固自己在软件行业的竞争力。

这时候问题来了，中小企业和创业公司没有那么大的财力怎么办？其实对于DevOps很多人理解不同，根据小编的理解以及对前段时间与IBM 大中华区技术总监 孙昕对DevOps交流的总结，给大家列出一二三，如有不同观点欢迎共同探讨。

DevOps为什么盛行

这个我们得从软件工程开始说起，从软件工程初期发展的独立主义，那时候的软件其实就是一个人独立开发，从需求到开发到测试，都一个人完成。因此那时候的软件很简单，功能单一，所以一个人完全可以。当从90年代开始强调了软件工程化至今完全进化成为一个产业链，软件工程应是由最早的需求、架构设计、开发、测试、发布一个完整的过程进化的时候，这时孙昕谈到了一个概念，那就是ALM。如何去理解这个ALM？就是把你的软件当成一个应用，这个应用就像人一样，有出生到成熟到死亡。官方专业的说法就是应用程序生命周期管理。但是时代在进步，我们的需求越来越大，软件的生产链条还要扩展，所以就是这次谈的DevOps这个概念。

为什么这么说？

在任何公司里，开发一个软件最终都是为了赚到钱，于是它有了大的商业规划，产品的需求，才有软件的开发这样的一个流程。那么软件再往上端走就直接到业务层，而软件的下一端理所当然是运维了。为什么这么说？如果在金融业，电信业，一出问题不及时解决就是个灾难。所以软件往下走一

定是运维，不断产生缺陷不断及时修复，所以现在的软件工程链条变得非常长，这就是DevOps这么个概念。

DevOps为什么这几年才开始提提及？用孙昕在IT客上的话说，就是有一个重要的技术瓶颈是这几年才突破的，在前几年一直在谈的一个概念叫JAZZ，翻译过来就是爵士乐，意思是需要协同起来才能奏出非常好的爵士乐。JAZZ的出现就是为了让软件真正的与底层全部打穿数据，让数据在说话。在之前还没有这个国际化标准的时候，但是今天我们能够看到了希望。

JAZZ的出现对整个行业包括软件，从业者等都会有着一定的改变。这个改变就是把程序员或者测试人员一直一来做的复杂且重复性的工作完全脱离出来，用更多的时间去做更有创造性的工作。小编记得孙昕说过这么一句话，如果没有JAZZ提及出来，就不会有今天的DevOps真正的出现。（详细请关注DevOps专题）

说到这里用一句话来感慨一下：软件从业者本身就是一个艺术家，不停地用创意改变着世界。

“土豪”的本质

如果从上面来看你对DevOps的概念有了了解，那么DevOps到底是用来解决什么问题呢？是解决公司的技术问题还是业务问题？

刚刚也说了，软件的上一端是业务端，任何一家企业开发软件，其实目的就是为了赚到钱。虽然说技术也是DevOps中的关键部分，但是从DevOps本质来说就是个就是个业务问题。不解决业务如何赚钱？光靠技术当然是不行的。

那么这个业务的流程是怎么组成的呢？从DevOps的活动组成部分中分为两种，第一是技术的驱动，第二是人为的驱动。比如说开发者，QA，架构，发布，安全，运维，其实都在这一流程中发挥了自己的作用。

无论是在传统行业或者软件研发这条工具生产线上做一些业务的规划，我们叫做产品线工程（PRE），在这些产品线本身其实就受到很好的工程化管理。从规划到实现，业务这部分已经占据了非常重要的地位。其实大家可以理解为项目管理，企业架构，数据架构，或者流程整合等等，做好哪些决策，投入到哪些大的项目中，这些都跟DevOps相关，实际上这些规划好了以后需求才会变成业务往前推，这样才能更好的获取你的需求是来自于支撑你的业务目标，对市场的压力做出快速，高效，经济，可靠的变化能力。小编认为，如果把业务抛开去谈DevOps，其实毫无意义。

也许有人会说，既然DevOps是业务流程的，为什么还要管它叫“DevOps”？

在早期，DevOps的出现就是为了解决开发与运维之间的问题，没有指明DevOps到底真是的范围，即使是解决问题到底多大，工程链条到底有多长。但是经过了在不断的实践过程中，大多的实践者就认识到面对市场压力的业务问题不解决，再怎么实践DevOps都没法真正意义上去解决企业上的问题。

所以要解决业务问题，我们就要从开发与运维两者之间的文化开始说起，这两者从诞生以来到现在已经存在着很大的冲突与脱节，每个企业的组织结构不同也会影响他们之间划分程度。要解决两者之间的脱节，所以你现在看到的谈DevOps最多的就是如何改善部署问题上，这是一种合理的选择。但是如果你说DevOps的出现

就是为了改善部署问题那就是误导别人了。

发掘“土豪”的共性

现在我们来谈谈DevOps的发展需要落实哪些事情，其实小编认为，新的话题出现都有一个共性特点，找到解决方案发展就越来越快。下面说3个共性：

1.标准流程统一化

回归到上面概念讲到应用程序生命周期管理，其实就是一个点对点的过程，那么在这个过程中不同阶段采取什么样的方法，这都需要建立起一套标准流程规范。这就得是国内外的土豪厂商们联合起来规划一个行业标准。

2.统一的工具使用

简单的说，一个大的团队，分布到世界各地的时候要做沟通和协调，你的第一反应是必须依赖工具，你必须花大量的金钱去购买工具，无论是利用云的技术，或者自动化部署，自动化发布，这都要依赖工具建立起来的桥梁下完成。

3.公司文化

我们都知道如果企业有心的技术出现，当你需要把其中某个部门合并了，那么他们之间就会造成很大的隔阂。这些隔阂可能会造成公司文化，工作习惯的改变，再者是一些角色的改变。如何让他们慢慢的整合在一起形成一条业务线，这也是一件不易的事。总而言之，不改变企业文化，就别谈DevOps。

所以，三者的依赖关系决定着企业实现DevOps的成败关键。

认识DevOps的落地

说到落地才是本文的关键，但小编的理解甚少，所以这里更多的借用孙昕的话来表达。

一个新兴技术或者方法的落地，其实我们不能用狭义的眼光去看它，广义才是它真正的本意，也是最有说服力的。

从狭义方面去讲的话其实DevOps已经落地，如果从广义来讲，目前DevOps还只是雏形。孙昕认为，落地现在主要是聚焦在开发和运维本身现在两部分，其实现在就已经有非常多的企业做到了，当然这只能说是市场的成熟度。在国外，特别是美国很多地方其实都已经有很多大的应用案例，但是在国内，据孙昕所说，据他观察，尤其是在国内的开发领域，我们经常比别人晚2-4年的时间。所以这就造成了为什么国外在实践，国内开始谈的局面。

想真正的落地你就要认识到：按时交付软件产品和服务，开发和运营工作必须紧密合作，只有才能更快，更安全，更准确的推进公司的业务，这才是DevOps的过程。

愿景：

最后总结的话其实对于DevOps也不好说，我们就来说说对它的愿景吧。

我们之所以能看到软件工程的将来，DevOps实际上就是软件从事于行业下一代发展的一个未来。那么所有以软件为驱动的创新将来是一个软件爆炸的时代，同时DevOps也将会是解决当代大数据上的移动互联，云计算转型的关键，将会更加驱动很多创新来改变人类的生活。■

将会改变未来IT世界的十种编程语言

■ 这里要说的都是革新，说这些的目的就是要保持关注最新技术。如果你是一个程序员，想要探寻未来技术，那这篇文章就是你的必读之选。我们这里列出了10种编程语言，10种将会改变IT世界工作方式的编程语言。这些语言已经在开始改变IT界的景象。

这里要说的都是革新，说这些的目的就是要保持关注最新技术。如果你是一个程序员，想要探寻未来技术，那这篇文章就是你的必读之选。我们这里列出了10种编程语言，10种将会改变IT世界工作方式的编程语言。这些语言已经在开始改变IT界的景象。看看吧:

Dart语言

这 种语言由谷歌制造，用来替代Javascript，弥补Javascript在web应用中大量使用时出现的缺点。对于Dart语言，谷歌的希望是，它将 成为web编程的新官方语言。它有着与C语言类似的语法和关键词。然而，一个跟Javascript的重大不同之处是，Javascript是以 prototype为基础的语言，可Dart里对象是用类和接口定义的，跟C++和Java一样。Dart语言还允许程序员将变量声明为静态类型。

Ceylon语言

这种语言被称为“Java杀手”。是由Gavin King(Hibernate 创始人，现任职于红帽)创造的，但他否认是在红帽(Red Hat)公司里开发的。Gavin King对Java的抱怨包括：罗嗦的语法，缺少一等函数(first class)和高阶函数(higher-order)，对元数据编程的支持很弱。特别的，他对缺少能够声明结构化数据定义的语法非常失望，他指出这使 Java只能“跟在XML屁股后面使劲”。Ceylon语言的目标就是要解决所有这些问题。

Go语言

这个大家应该都知道 了，谷歌创造了一个叫做Golang或Go的编程语言。据一些技术分析家说，它将最终完全替代Java。这是一种通用型的语言，可以用来开发任何软件—— 从普通应用到系统编程。虽然这种语言还不成熟，各种语言特征和规格还在变化，但程序员如今已经用它来开发工作了。

F#语言

这 种语言已经在计算机科学研究和学术界里流行很久了。F#(发音是“F-sharp”)，是一个微软制造的语言，设计时既考虑了功能性又考虑的实用性。因为 它是一种可以运行在.NET通用语言运行环境(CLR)的一等函数(first-calss)语言，它能跟其它CLR语言(如C#和VB)一样可以访 问.NET平台上的所有程序库和功能特征。

Opa语言

Web开发太复杂。即使一个简单的web应用，也会包含有多种语言 交织的无数代码：客户端有HTML和Javascript，服务端有Java和PHP，数据库里有SQL，等等。Opa语言并不是来替代其中的某个语言。 事实上，它是想一次把这些语言全消灭掉——通过倡导一种全新的Web编程模式。在一个Opa应用中，客户端UI，服务端逻辑，数据库I/O，全部由一种语 言实现——Opa语言。

Fantom 语言

你是否开发过Java或.Net应用？如果使用Fantom开发，你可以选择 使用它们任何一种平台，甚至中途切换平台。这是因为Fantom语言专门是为跨平台移植设计的。Fantom工程不仅包括一个可以输出JVM和.NET CLI字节码的编译器，还包括一套从Java和.Net中提取的API，从而可以创建一个额外的可移植代码层。

Zimbu语言

这种奇特的语言从其它各种语言中吸取元素和成分，它是Bram Moolenaar的 智慧结晶。Bram Moolenaar是Vim文本编辑器的缔造者。这种语言被规划为要快，简洁，可移植，易读。它的语法独特、与众不同，但功能丰富。使用C语言风格的表达 式和操作符，但有自己的关键字、数据类型和块结构。它支持内存管理，线程，管道。可移植是它的核心理念。尽管Zimbu是一种编译型语言，但Zimbu编 译器输出的是ANSI C代码，这样可以让本地的C编译器来把它编译成本地平台的二进制代码。

X10语言

这是一种并行处 理语言，曾经只是用在特定领域里的软件开发。然而，随着多核CPU和分布式计算的普及，今天的其它编程语言都似乎跟不上这种趋势的步伐。这就是为什么 IMB研究机构开发了X10语言——一种专门为现代并行架构设计的语言，目标就是要把开发效率提高“10倍”。X10语言的并行能力来自使用分块全局寻址空间(PGAS)编程模式。代码和数据被分割成小的单元，分布到一个或多个“空间”，使得将一个单线程程序升级成运行在多核处理器上的多线程程序变得简单。

haXe语言

haXe(发 音是“hex”)胜过任何一种可移植的编程语言。它是一种可以应用到多种操作环境的多平台语言——从本地二进制到脚本解释器到虚拟机。程序员用它开发出代 码，然后编译成目标代码，JavaScript，PHP，Flash/ActionScript或NekoVm字节码等。

Chapel语言

对于应对当今世界高性能计算的特性，Chapel是一种出色的编程语言。这种语言专门为超级计算机和集群设计的，它是Cray(超 级计算机之父)的Cascade研究课题的一部分，由美国国防部高级研究计划局(DARPA)参与启动，有一个宏大的高性能计算设想。Chapel语言的 语法有很多源头，除了常见的如C，C++，Java外，它还借鉴了一些科学研究性语言(比如Fortran和Matalb)里的概念。它的并行处理特征是 受ZPL和High-Performance Fortran的影响，这些语言也都是Cray的早期研究项目。■

用来理解Java编程语言的8个图表

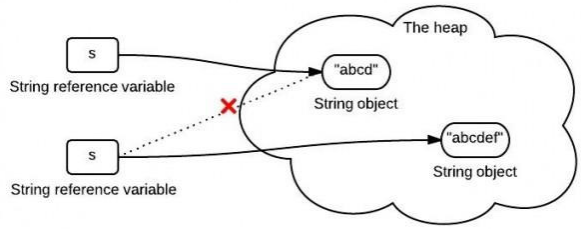
开源中国 /译

很多时候，一张图比你说 1000 个字能更有的说清楚一个问题。我们列举了 8 个关于 Java 言的图表，或许可以让你对 Java 有着更深入的识。

1. 字符串不变性(String Immutability)

下面的图表显示执行如下两行代码所发生事：

1. String s = "abcd"; s = s.concat("ef");
2. string-immutability

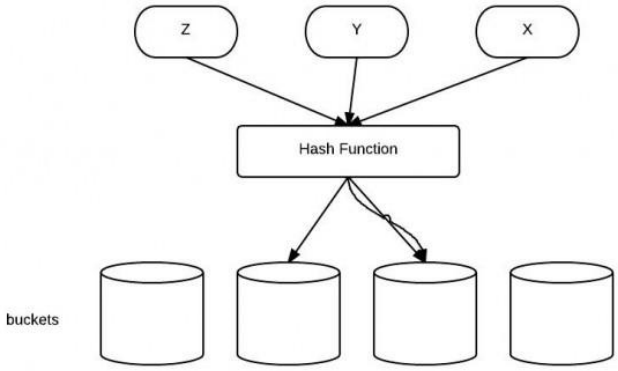


2. equals() 和 hashCode()

HashCode 方法设计用来提升性能，hashCode 和 equals 之间的差异有：

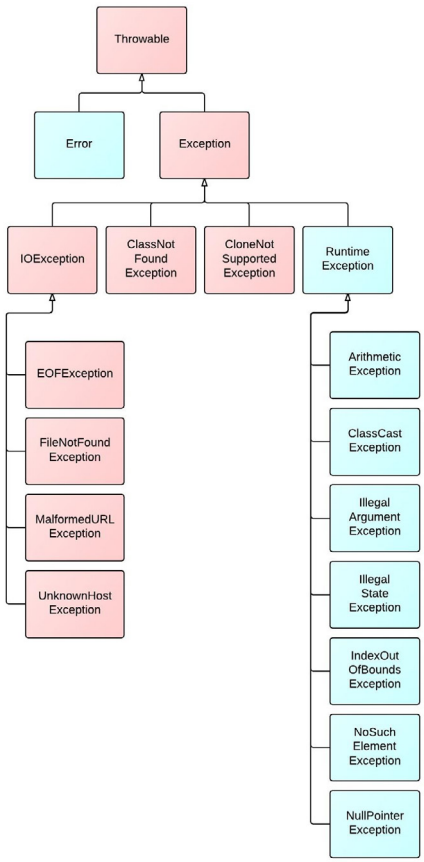
1. 如果两个对象是 equal 的，那么他们必须有相同的 hashCode

2. 如果两个对象有相同的 hashCode，但他们可以是不 equals 的



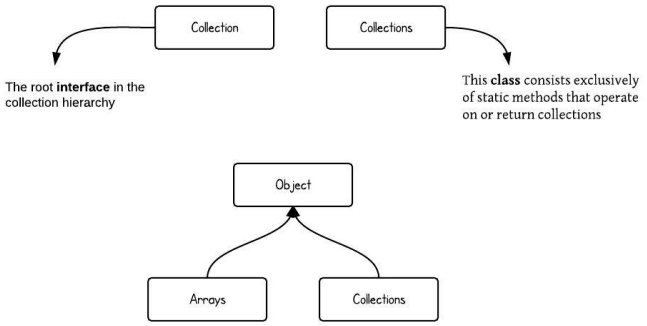
3. Java 异常类层次

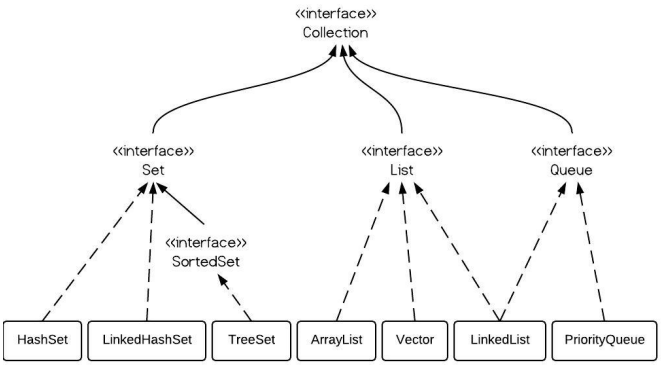
红色的方块为 Checked Exception，必须被捕获或者是在方法中使用 throws 声明抛出。



4. 集合类层次

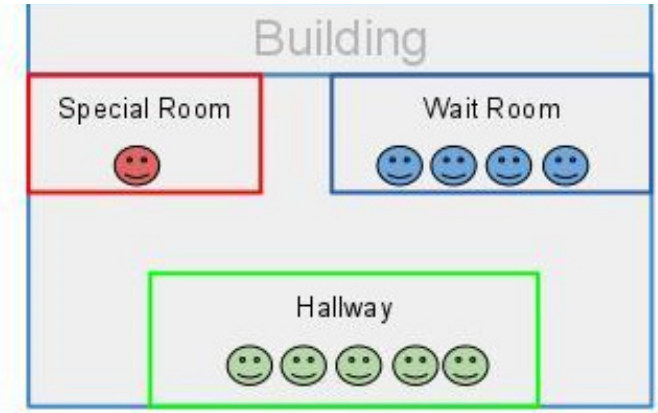
注意 Collections 和 Collection 之间的差别。





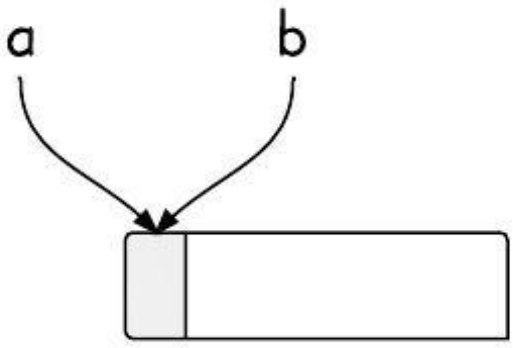
5. Java 同步

Java 同步机制可以通过如下比喻来说明



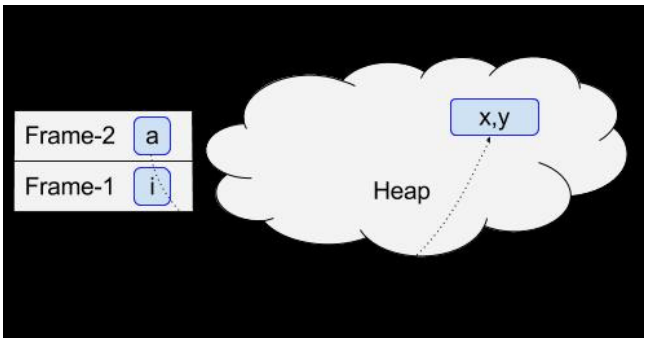
6. 混淆 Aliasing

混淆意思是有多个别名指向同一位置，而且这些别名有着不同的类型

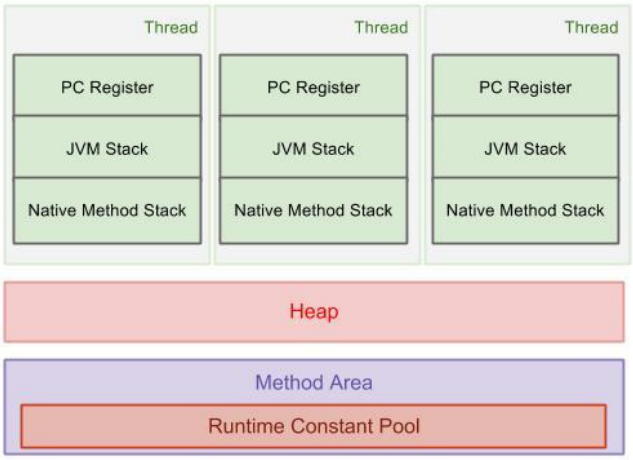


7. 栈和堆

该图标显示方法和对象在运行时内存中的位置

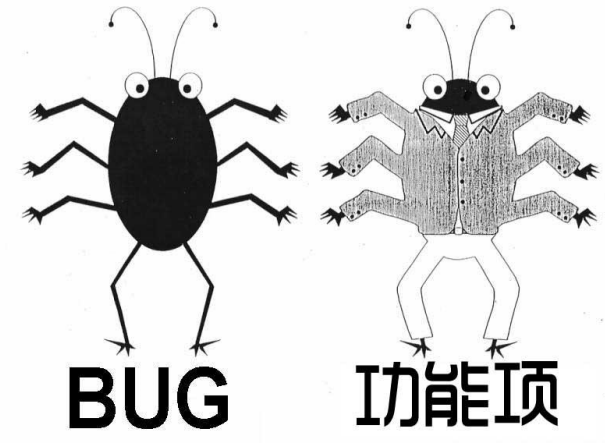


8. JVM 运行时数据区域 Run-Time Data Areas



■

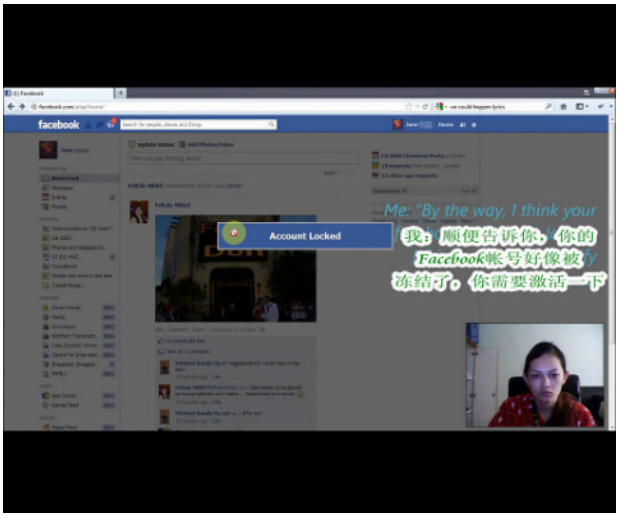
开心



下面是视频的截图。

完整视频请进入下面的链接观看

http://www.tudou.com/programs/view/7uwokEK5lTg/?resourceId=0_06_02_99



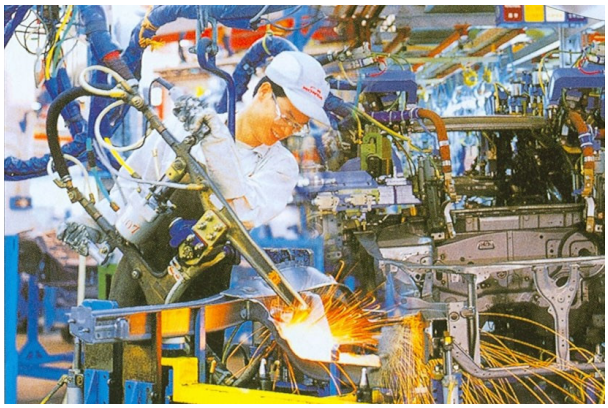
像程序员那样去求婚

不知大家是否还记得，在《一个老程序员的建议》这篇文章的最后一句话是这样写的：

你会编程。他们不会。这真他妈的酷毙了。

几乎有一半的评论对这句话表示了高度的欣赏。的确，这句话并不是被人们喜欢称做“nerd”的程序员聊以自慰的话。懂编程，懂计算机，在这个计算网络无处 不在的社会里已经成为了一种得天独厚的优势。网上抢火车票，抢小米订单，这是懂得web技术的人的长项。手机越狱，翻越防火墙，盗用Wifi，这些是懂加 密解密技术的程序员的发挥才能的地方，等等。而下面将要向大家介绍的一个视频，更是只有程序员才能实现的事情。视频中的这个小伙伪造了一个 Facebook网站，然后让她的女朋友误以为她的Facekook被锁住了，她就开始在这个假的Facebook网站上进行操作，结果却发现了一个惊喜……

这个视频里的主题音乐非常好听，它是由AJ Rafael演唱的《We Could Happen》，你可以在这里看到它的完整MTV。■



谁说设计师不会写代码？Photoshop脚本语言简介

> 今天，我们将介绍给你一种高级的自动 化技巧：脚本语言。所有的这一切仅仅需要你有一点点关于JavaScript的基本知识，这对于我们中的一些网页设计师往往都是具备的。

自动化对每个设计师的工作来说是很有用的。它可以在重复的任务上节省宝贵的时间，还能够帮我们更快捷、更容易的解决一系列问题。

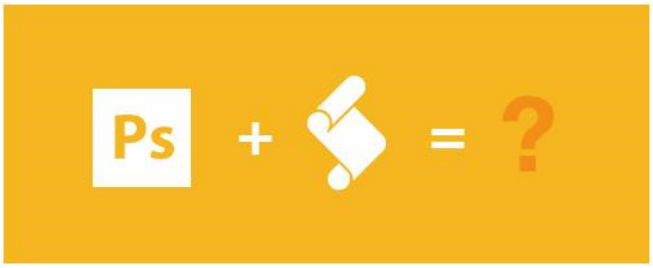
你可以使用photoshop的动作来使工作流程自动化，这是很流行的，大多数人都知道并且已经在使用的方法。今天，我们将介绍给你一种高级的自动 化技巧：脚本语言。所有的这一切仅仅需要你有一点点关于JavaScript的基本知识，这对于我们中的一些网页设计师往往都是具备的。

我很多年前就知道Photoshop的脚本语言，但是我几个月前才开始决定研究它。我忽视了它是因为我认为那是聪明的具有数学思维能力的程序员的领域。但是我错了，今天我将要来告诉大家，尽管它需要一些基本的程序技巧，脚本语言并不是那么难掌握。

但是，一开始，我们得回答以下这几个明显的问题：

为什么我们需要脚本语言？

为什么我们在Photoshop已经有了很棒的动作之后还要需要学习脚本语言？答案是交互性！当你使用动作时，你不能真正的控制它在不同的情况下的表现，就像录像带不停的一遍又一遍的播放而没有任何改变。



一个脚本语言更灵活，它表现形式的改变取决于你输入的参数或者是应用程序的内容。听起来很有用？不是么？

要求

你不需要是一个会写脚本语言的高级的程序员。我就仅仅是一个平面设计师，就像你们大多数人一样。但是你需要对JavaScript至少有一个基本的了解,以及一些属性和方法的经验去领悟这篇文章的大部分内容。

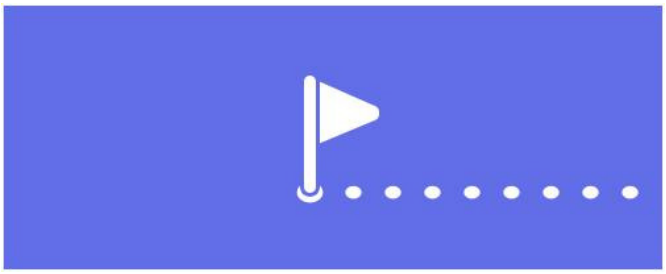
如果你对JavaScript一点都不了解，别害怕！

有很多地方你可以了解程序的一些基本知识。例如：Codecademy，有很棒的完整的交互式课程。

我使用CS5，但是这篇文章中讲的也同样适用于新版本。Adobe从CS5开始已经没有对其脚本API进行更新。我会选择看最新的脚本文档版本，虽然它是cs6的.

开整

当你开始在PS中记录动作的时候，你设定了一个达到某一结果的步骤顺序，这就是你的算法。然后你按下开始动作，在PS里面就一个接一个的重复你刚才 的动作。脚本也差不多，但是不会一步一步的在PS中完成，而是你把它们一条一条的写出来。在PS里面大多数你想要完成的动作都有相同的功能按钮可以完成。



如果你需要创造一个动作将你的文件从原始大小放大到150%，你需要完成以下几步：

1，打开图像-图像大小

2，长宽输入150%

3，点击确定。

同样的脚本语言会是这样写：

1，为这个应用命名为：app

2，选中文件：activeDocument

3，把这个属性命名为重设图像大小：resizeImage(width,height)

代码就是这样：

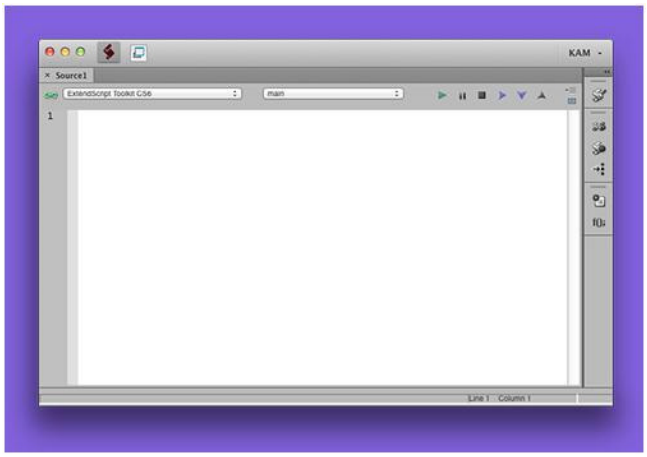
```
1. app.activeDocument.resizeImage(“150%”,
   “150%”);
```

语言

有三种方式在PS里面写脚本：在mac上用AppleScript，Windows用VBScript，或者用JavaScript在两者上都可以。我使用第三种方式，因为它可以跨平台，并且我有一定JavaScript的基础。

工具

Adobe有它自己的写脚本的工具，叫做：ExtendedScript Toolkit.

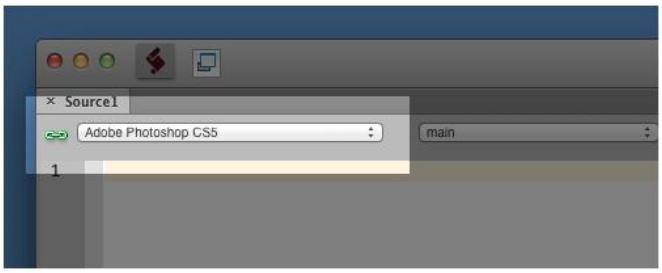


这个工具包在PS里面，你可以在这个文件夹中中找到：

1. Mac OSX: /Applications/Utilities/Adobe Utilities CS6/ExtendScript ToolkitCS6/
2. Windows: C:\Program Files\Adobe\Adobe Utilities-CS6\ExtendScriptToolkit CS6(64位的Program Files(x86))

ExtendedScript Toolkit的用户界面非常的

简单。开始写代码，第一步就是要在下拉菜单中选择目标应用。如果PS已经在运行，就可以看下下拉菜单旁边的绿色链接图标：



这个时候，你可以像这样写：

```
1. alert(“Hello Photoshop!”);
```

ExtendedScript Toolkit 有一些其他的调试代码的不错特性，但是这一段代码就这样就够了。你可以通过：帮助-JavaScript Tools Guide学到更多如何使用它。

你可以使用任何文本编辑来写代码，只需要保存为.jsx格式文件就好了。你必须在PS里通过File-Scripts-Browse来找到并运行它。 或者是，在PS里面打开脚本文件。你也可以在脚本的前面加上一行代码，这样这个代码就会常在PS里面打开：

```
1. #target photoshop
```

直接保存你的代码在 Photoshop/Presets/Scripts/，然后通过File-Scripts访问它们。你也可以设定一个快捷键，前往 Edit-Keyboard Shortcuts，链接到 File-Script-[你的代码名称],然后选择一个你想要设定的快捷键。

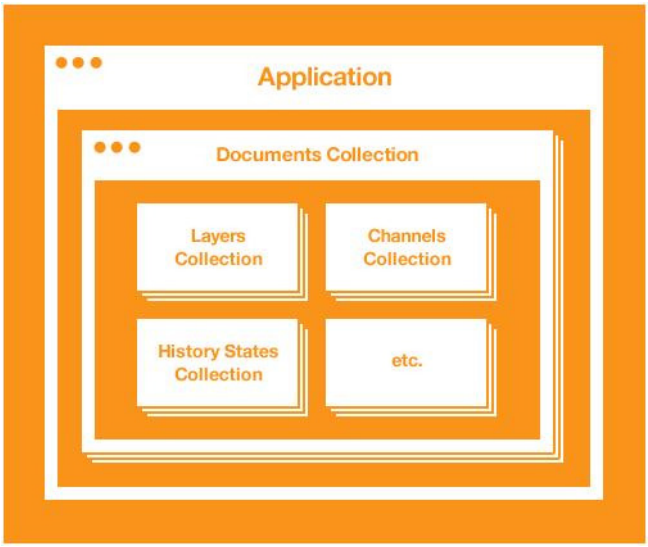
ExtendedScript Toolkit可以在整合的开发环境下运行和调试代码，同时它还有一个目标模型指示器来安装，这是很有用的。所以我推荐使用toolkit来写脚本。不幸的是，Mac版本的有时候会崩溃，所以要记住这一点。

Photoshop对象模型

为了使脚本写起来简单些，你需要懂得在Photoshop文件对象模型（DOM）中事件是怎样互相联系的。如果你观察PS本身，理解起来并不困难。PS的DOM里面最主要的对象就是应用程序。在应用程序里面，我们有一个文件夹在PS里面是当前打开的。

每一个文件包含一些元素：例如图层（被称为Artlayers）,图层组（layerSets）,通道，历史记录等等 - 就像一个普通的PSD文件。

下面就是一个简单的可视化的PS的DOM。更详细的包含层级的可以在”Adobe Photoshop CS6 Scripting Guide”Pdf文件中 1 2 页里找到。



一个简单的可视化的Photoshop DOM

这里面的每一个目标都有它自己的属性和方法你可以编辑它。例如，在一个文件中改变所选图层的透明度，你就可以前往Application-Document-layer-Opacity然后选择你的期望值。代码就是这样写：

```
1. app.activeDocument.activeLayer.opacity = 50;
```

你应该可以猜到，activeDocument和activelayer决定了当前选择的文件和图层。

你可以在“Adobe Photoshop CS6 JavaScript Scripting Reference”PDF文件中找到大部分的目标和它们的属性和方法的说明，或者在ExtendedScript Toolkit中通过前往help-object Model Viewer.

让我们来看看在一个真实的例子中是如何运行的。在接下来的段落中，我们将会基于一个动作写一段我们自己的代码。

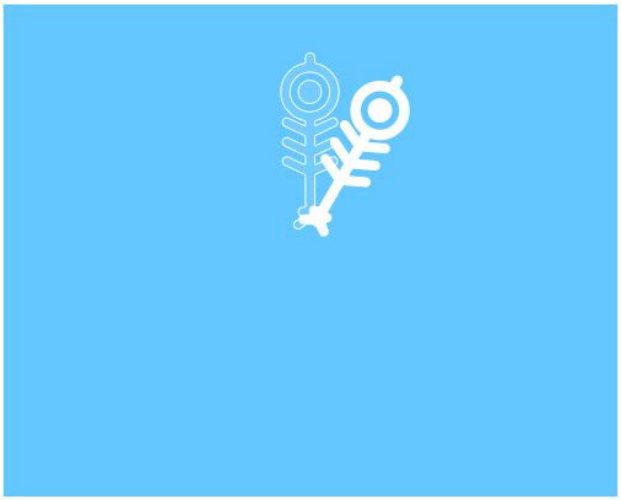
用代码来重现自我旋转动作

几年前的圣诞节，我有一个想法就是用动作帮助我画一个雪花。

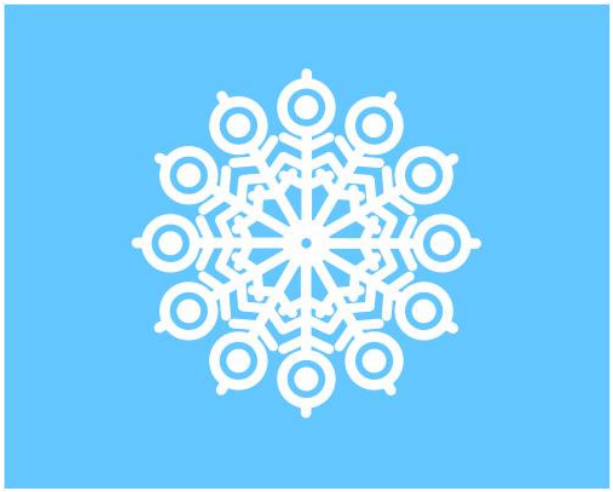
画雪花

1，首先画一支雪花的图案。

2，复制这一支，同时旋转一定的角度。



3，重复第二步直到一个完整的圆。



手动去复制和旋转每一个元素非常麻烦，所以我想出了一个自动的动作(an action to automate it)去完成它。算法是这样的：

1，复制元素。

2，使用变化工具按照你所选择的角度去旋转这个元素。

3，复制图层。

4，使用“重复变换”功能。

5，重复动作4和5直到一个完整的圆。

非常不错！但是这个动作有一个缺点：根据你在第三步设定的角度数值的算法，你只能设定一定数量的雪花的分支。

回到当我还不是很熟悉脚本的时候，我做了几个版本的动作，每一种产生的雪花都是不同的分支数。

今天，我们将用你输入分支的数值的动态脚本来重绘这个动作，让我们开始吧。

算法

当你开始写脚本的时候，在挖掘代码本身之前先设定算法是个不错的主意。在我们的情况下，算法将会是这样的：

1，询问用户输入分支的数目。

2，计算旋转的角度。

3，通过第一步设定的数目来复制并旋转图层。

首先让我们从把当前或选定的图层作为变量保存起来，为了将来使用：

```
2. // Save selected layer to variable:
3. var originalStem = app.activeDocument.
   activeLayer;
```

在JavaScript 中注意，你可以标记两条双斜线(//)做注解。注解被用来为未来相关部分的代码做解释但是不影响脚本的运行。

现在让我们回到我们的算法上。

1，要求用户输入

我们通过prompt(message,defaultvalue[,title])这个功能来获取用户的输入信息：。这个功能表明一个有 着”message”对话框和一个包含这”fefault value”的输入框。当用户点击“确定”，这个给你功能就回到输入值；因此，我们需要保存它为一个可是用的变量。

```
1. // Ask user for input by showing prompt box
   and save inputted value to variable:
2. var stemsAmount = prompt(“Processing
   \”” +originalStem.name+”\”\nHow many
   stems do you need?”, 12);
```

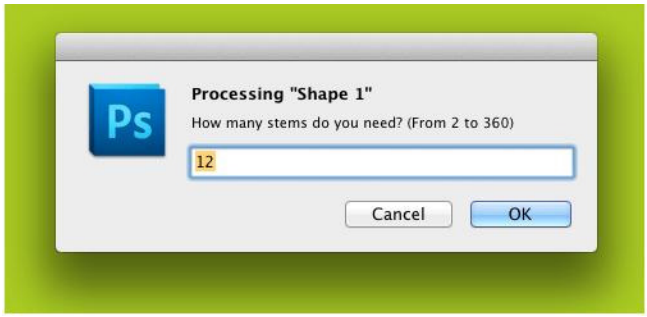
注意我使用了“orginalStem.name”在这段信息里面。所以对话框会现实所选择图层的名称。

在Mac OS X中，在信息中的第一行是宽的，作用是标题。因此，我们主要的信息应该在第二行。另起一行，输入“\n”.

在Windows中，你可以在功能中指定第三种参数来设定一个标题：

```
1. // Ask user for input by showing prompt box
   and save inputted value to variable:
2. var stemsAmount = prompt(“How many
   stems do you need?”, 12, “Processing
   “+originalStem.name);
```

如果我们在PS中运行这个代码，将会看到这样一个对话框：



当用户点击“确定”，输入值将会保存到

stemsAmount变量中。如果用去点击“取消”，这个功能将会返回一个无效值。这个我们后面要使用到。

2，计算旋转的角度

为了计算旋转的角度，我们需要通过分支的数目来分360度（一个整圆）：

```
1. // Calculate the rotation angle
2. var angle = 360 / stemsAmount;
```

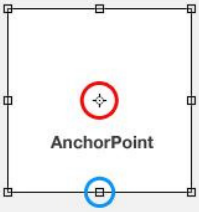
3，复制和旋转

现在我们已经有了我们需要复制的分支的一切。为了这样去做，我们将使用“for”循环。它可以让我们按照我们想要的次数来反复的运行一段代码。我们的循环将会是这样的：

```
1. for(var i = 1; i < stemsAmount; i++){
2.     // This code will run “stemAmount - 1” of
   times
3. };
```

注意第一个在程序中的对象例子已经有了值为0，但是因为我们第一个图层已经在画布上了，我们从1开始这个循环。

为了辅助和旋转我们的图层，我们将会使用：duplicate()和rotate(angle,AnchorPosition)函数：在angle 里面被旋转图层的数目通过复制的指数相乘而得。 Anchorposition决定了哪个图层将会旋转的点。当你在PS中使用旋转工具的时候你可以看到这个点—它看起来是一个小小的加了十字的圆圈。在脚 本中，它仅有9个指定的值-i.e.9个位置的锚点：



在我们这个情况下，它是这个图层按钮的中心。BOTTOMCENTER. PS在这里或那里的一些功能上使用了很多其他的一些常量，你可以在”Adobe Photoshop CS6JavaScript Reference”PDF文件中的197页找到。因此我们的循环就是这个样子：

```
1. // Duplicate and rotate layers:
2. for(var i = 1; i < stemsAmount; i++){
3.     // Duplicate original layer and save it to the
   variable
4.     var newStem = originalStem.duplicate();
5.
6.     // Rotate new layer
7.     newStem.rotate(angle * i, AnchorPosition.
   BOTTOMCENTER);
8. };
```

完整的代码就是下面这个样子，你可以试着运行：

```
1. // Save selected layer to variable:
2. var originalStem = app.activeDocument.
   activeLayer;
3. // Ask user for input by showing prompt box
   and save inputted value to variable:
4. var stemsAmount = prompt( “Processing
```

```
\”” +originalStem.name+”\”\nHow many stems do you need?”, 12);
5.
6. // Calculate the rotation angle:
7. var angle = 360 / stemsAmount;
8.
9. // Duplicate and rotate layers:
10.     for(var i = 1; i < stemsAmount; i++){
11.         // Duplicate original layer and save it to the variable
12.         var newStem = originalStem.duplicate();
13.
14.         // Rotate new layer
15.         newStem.rotate(angle * i, AnchorPosition.BOTTOMCENTER);
16.     };
```

最后的润色

我通常会试着使用脚本来完成我的主要目的。当一切都正确的运行起来的时候，我将会开始优化代码。在我们这种情况下，我们需要确保用户在提示框中输入一个有效的数值—i.e.一个正整数，而且要大于1。

当然，为了不让PS疯掉，我们会限制分支的书目，我们规定，小于100.为了这么做， 当他们提交了一个无效的数值的时候 ，我们需要使用一个“while”循环来告诉用户一个错误的信息。而且这个提示框将会一直存在，直到用户输入一个有效值或者点击“取消”按钮。（记住如果 用户点击取消将会提示无效值）。

新的代码将会是这样的：

```
1. // Save selected layer to variable:
2. var originalStem = app.activeDocument.activeLayer;
3. // Ask user for input by showing prompt box and save inputted value to variable:
4. var stemsAmount = prompt(“Processing \”” +originalStem.name+”\”\nHow many stems do you need? (From 2 to 100)”, 12);
5.
6. // Check that user entered a valid number and, if invalid, show error message and ask for input again
7. while(isNaN(stemsAmount) || stemsAmount <= 0 || stemsAmount > 100){
8.     // If user clicks “Cancel” button, then exit loop
9.     if(stemsAmount == null) break;
10.         // Show error message…
11.         alert(“Please enter number in range from 2 to 100”);
12.
13.         // …and ask for input again
14.         stemsAmount = prompt(“Processing \”” +originalStem.name+”\”\nHow many stems do you need? (From 2 to 100)”, 12);
15.     };
16.
17.     // Run the copying process
18.     if(stemsAmount != null){
19.         // Calculate the rotation angle
```

```
20.         var angle = 360 /
  parseInt(stemsAmount);
21.         // Duplicate and rotate layers:
22.         for(var i = 1; i < stemsAmount; i++){
23.             // Duplicate original layer and save it
  to the variable
24.             var newStem = originalStem.
  duplicate();
25.             // Rotate new layer
26.             newStem.rotate(angle * i,
  AnchorPosition.BOTTOMCENTER);
27.         };
28.     };
```

你可能注意到：我们使用了“isNaN(value)”这个功能，它返回“true”如果“value”不是一个数字，同时当我们计算旋转的角度的时候，“parseInt(value)”把“value”转换成一个整数。

接下来我们要做的事情是管理图层，通过为它们增加一个索引来重命名我们的图层。同事也要确保我们不会把文件的图层搞乱，让我们把我们的分支编组。

为图层重命名不是一个很难的事情。我们只需要使用图层的“name”属性，然后为它们增加一个索引数字：

```
1. // Add index to new layers
2.  newStem.name = originalStem.name + “ “ +
  (i+1);
```

PS应用程序界面里的编组被称为“LayerSet”,我们通过“layerSets”属性可以进入文件的所有编组。为了给文件增加一个新的组，我们需要称“layerSet”方法为“add()”：

```
1. // Create a group for stems
2. var stemsGroup = app.activeDocument.
  layerSets.add();
3.     stemsGroup.name = originalStem.name +
  “ (“+stemsAmount+” stems)”;
```

然后，为了把一个图层增加到组里面，我们会使用“move(relativeobject,ElementPlacement)”函数。请注意，“move()”函数只是把图层移动到图层堆，而不是移动到画布上。（你可以用“trabslate(deltaX[,deltaY])”函数把图层移动到画布上）

ElementPlacement是另外一个常量，这个常量决定我们怎样把的图层跟 relativeobject 关联在一起。在我们的案例里，我们使用 ElementPlacement.INSIDE 把一个普通图层放进一个组里面：

```
1. // Place original layer in group
2. originalStem.move(stemsGroup,
  ElementPlacement.INSIDE);
```

我们使用 ElementPlacement.PLACEATEND.把每一个拷贝的新图层放在所有图层租的底部。结果就是我们的所有图层都是以上升的顺序排列，第一个图层在顶部，最后一个图层在底部：

```
1. // Place new layer inside stems group
2. newStem.move(stemsGroup,
  ElementPlacement.PLACEATEND);
```

你可以在”Adobe Photoshop CS6 Scripting Guide”PDF文件中202页里，找到更多关于“ElementPlacement”的内容。■

为什么C++程序员不想改用Go语言

■ 这是一个私人谈话。我不单是对在这坐的Go开发团队成员说，我要感谢团队在推动Go发展上所做的一切。我还想感谢Go SF组织者给了我跟大家交流的机会。

下面是我在2012年六月旧金山Go SF会议上的发言。

这是一个私人谈话。我不单是对在这坐的Go开发团队成员说，我要感谢团队在推动Go发展上所做的一切。我还想感谢Go SF组织者给了我跟大家交流的机会。

几个星期前我被问道这个问题：“你被鼓励转到Go后遇到最大的惊喜是什么？”。我立刻知道了答案：虽然我们预期C++程序员会将Go当做一个替代者，然而转到Go的程序员更多来自于如Python和Ruby等语言，很少有来自C++。

Ken、Robert和我，当我们还是C++程序员时我们设计了一种新的语言来解决我们认为需要用这门新语言来解决的问题。这好像是自相矛盾的，其他C++程序员并不在乎。

下面是我在2012年六月旧金山Go SF会议上的发言。

这是一个私人谈话。我不单是对在这坐的Go开发团队成员说，我要感谢团队在推动Go发展上所做的一切。我还想感谢Go SF组织者给了我跟大家交流的机会。

几个星期前我被问道这个问题：“你被鼓励转到Go后遇到最大的惊喜是什么？”。我立刻知道了答案：虽然我们预期C++程序员会将Go当做一个替代者，然而转到Go的程序员更多来自于如Python和Ruby等语言，很少有来自C++。

Ken、Robert和我，当我们还是C++程序员时我们设计了一种新的语言来解决我们认为需要用这门新语言来解决的问题。这好像是自相矛盾的，其他C++程序员并不在乎。

今天我想来说说是什么激发我们创建 Go 语言，以及为什么得出这样的结论应该是意料之中的事情。我承诺这将是一段更倾向于 Go 语言而不是 C++ 语言的发言，所以即使你不了解 C++ 语言，你也可以从这篇文章中了解一二。

实际上，这篇文章的结论可以概括如下：你认为 less is more （少即多），还是 less is less（少即少）？

这里我用一个真实的案例作为比喻。Bell 实验室中心原来是用 3 位数字来命名的：111 是物理研究所、127 是计算科学研究所等等。在 20 世纪 80 年代，随着我们对研究的深入了解，已经不足以用 3 位数字来简单地命名研究所了。所以，我们的研究中心编号变成了 1127 。 Ron Hardin 半开玩笑地说如果我们真的了解了这个世界，我们可以去掉编号前面的 1 位，从 127 变为 27。当然，管理层不会采纳这个玩笑，也并没有被期望他们这么做。但是，我认为，Ron 的说法是有道理的。Less can be more （精简反而有更深的内涵）。

你了解得越深入，你就能够用更精炼的语言来概括。

记住这个道理。

回到 2007 年 9 月，我正在做一些为一个大型的 Google C++ 项目做一些琐碎而中心的工作。你可能接触过这样的程序，在我们大型分布式编译集群上编译这个程序就用了大概 45 分钟。后来，听说了在 C++ 标准委员会的 Google 工程师将会进行 C++0x（即现在的 C++11）新特性的演讲。

在这场 1 个小时的演讲里，我们听说了 C++0x 计划中的 35 项新特性。实际上，或许还有更多的，但演讲中只描述了其中的 35 项。当然，一些新特性可能很微小，但是演讲中的特性却都十分重要。一些特性非常精妙而难以理解，像右值引用（rvalue-reference）；但同时 也有很多 C++ 特色语法，像可变参数模板（variadic templates）；而其他一些则非常疯狂，像用户定义数据标志（user-defined literals）

此刻我问了自己一个问题：C++委员会真的相信C++因为没有足够的特性而出问题吗？当然，作为Ron Hardin玩笑话的变种，简化编程语言相比于增加语言是更大的进步。这想法是有些可笑，但先记住它。

在那次C++演讲前几个月，我曾给自己一个演讲，你可以去YouTube上看，内容是关于我在1980年间创造的一个娱乐性质的并发语言。那个语言叫做Newsqueak，当然它是Go的前驱。

我做那个演讲是因为Newsqueak中的一些想法是我在Google工作时错过的，而我又重新对这些想法进行了思考。我确信它们将使得写服务器代码更加简单，而且Google将会从中受益。

事实上我曾经尝试去寻找一个方法把这些想法引入C++，但是失败了。把并发操作和C++的控制结构连接起来太难了，反过来这样也很难看到真正的优势。再加上C++让自己看起来非常难处理，尽管我承认很难使语言真正易用。所以我遗弃了这个想法。

但C++0x演讲让我重新思考。真正困扰我的——我认为也困扰着Ken和Robert——是具有原子类型的新C++内存模型。在一个已经超负荷的类型系统 中加入这样一个定义微观细节的集合，让人感觉是错的。这看起来也非常短时，因为硬件很可能在下一个十年里有明显的变化，把语言和今天的硬件结合这么紧密将 是不明智的。

会议结束后，我们回到办公室。继续开始了新一轮的编译，然后转过桌子面对着 Robert，开始询问那些关注的问题。在编译结束以前，我们说服了 Ken ，决定自己动手做点什么。我们并不希望一直用 C++ 写程序，我们，特别是我，希望在为 Google 写代码的时候能够自然地使用并发编程技术。我们也希望强调“Programming in large”的方向。

我们在白板上写了很多我们想要的特性。我们从大方向出发，忽略详细的语法和语义只关注事物的主要部分。

我还保留着在那周里一个邮件列表，下面是一些节选：

Robert：起点： C语言，解决一些明显的瑕疵、删除杂质、增加一些缺少的特性。

Rob：命名为 Go 语言吧，你可以为这个名字想一些原因。但是它已经有不少美好的属性了。简单地说，易于书写。工具命名也容

易：goc、gol、goa。如果有一个交互的调试器、解释器的话，它可以就叫“go”。文件名后缀就是“.go”。

Robert：空接口：interface {}。可以被所有接口实现，所以，这个接口可以代替 void* 的位置。

这时候，我们并没有完全弄清楚这种语言。举例来说，我们用了大概一年时间才弄清楚 array 和 slice 的结构。但是在最初的那几天，很多重要的特性就已经出现了。

注意到 Robert 说 C 语言是起始点，而不是 C++ 语言。我不确定，但我相信他指的是 C 语言，因为当时 Ken 也在。但是实际上，最后我们并不是真的从 C 语言开始扩展。我们完全重新开始了这门语言，稍微从 C 语言借鉴了操作符、括号以及一些普遍的关键字。（当然，我们也从其他我们所知道的语言中借鉴了很多想法。）在任何情况下，在我看来，我们从基础上、打破一 切的思路以及重新开始来回应 C++ 语言。我们并不是尝试设计一个更好的 C++ 、甚至说一个更好的 C 语言。它只是一个对我们所关注的软件更好的语言。

最后，它当然和 C 或者 C++ 都相当不同。不同点甚至比大家注意到的多。我列举一下 Go 语言对 C/C++ 语言的关键的简化：

- 正则语法（不需要一个符号表来做语法分析）
- 垃圾回收
- 没有头文件
- 没有循环依赖
- 常量只是数字
- int 和 int32 是两种不同的类型
- 用大小写字母区别变量的可见性
- 对所有类型可用的方法（没有类的语法）
- 没有子类型继承（没有子类）
- 包层次的初始化和良好定义的初始化顺序
- 文件的编译以包组织
- 任意顺序出现的包层次全局变量
- 没有算术转换（arithmetic conversions）（通过常量支持）
- 接口是隐含的（不需要用“implements”显式声明）
- 嵌入的（没有超类的好处）
- 方法被声明为函数（不需要特殊的地方）
- 方法就是函数
- 接口是没有数据的方法
- 方法以参数名匹配，而不是通过类型
- 没有构造函数或者析构函数
- 后增量运算（x++）和后减量运算（x--）是语句而不是表达式
- 没有前增量运算和前减量运算
- 赋值操作不是表达式
- 没有序列点（sequence point）的概念：evaluation order defined in assignment, function call
- 没有指针运算
- 内存分配总是初始化为 0

- 可以获取本地变量的地址
- 在方法中没有 this 关键字
- 分段的栈结构
- 没有 const 或者其他类型注解
- 没有模板结构
- 没有异常处理结构
- string、slice 和 map 是语言结构
- 数组越界检查

尽管需要精简以及缺失部分的列表依然很长，但我相信，Go 比 C 或 C++ 更具表现力。积少可以成多。

当然，没有人可以一下子拿出所有的东西。诸如类型行为、实践中运行良好的语法以及使库间相互运行良好的不可言喻的部分等等需要慢慢积累。

我们也添加了一些 C 或 C++ 中所没有的东西，例如 slices 和 maps、复合声明，顶级文件表达式（非常庞大，且多数部分尚不为人知），反射，拉圾回收等。当然，还有原生的并发。

明显缺失的一项是类型层次。这里我可以说脏话么?

早在GO语言第一次出现的时候，就有人说，不能想象在工作中如何使用一个没有泛型的编程语言。这是我在某处作报告的时候，知道了这种奇怪的想法。

公平的说，他只是以自己的立场发表看法，或许他真的很喜欢C++中的STL为他带来的便利。所以为了讨论的目的，我们只有把他的想法看成很肤浅表面的东西。

按照那种想法，编写一个容器，例如整型的list和字符串的map，是一种不能承受的工作量。我知道了这种奇怪的想法后，花了我很少很少的编程时间实现了这些容器，即使是使用一些没有泛型的编程语言。

但更重要的是，那种想法中把类型作为解决这一难题的方法。类型，不是多态函数，不是语言基元不是其他方法的辅助，竟然是类型。

那是与我相关的细节。

从C++和Java转向Go的程序员不懂得使用类型编程的想法，尤其是继承、子类和与之相关的内容。也许关于类型我是个门外汉，但我还没发现一个模型能有如此的表现力。

我的一位老友Alain Fournier曾对我说，他把学术工作的最低层次看作是分类学。你知道吗？类型分层恰恰就是分类学。你需要决定哪块进什么箱子，每个类型的父类，无论是 A继承于B还是B继承于A。一个可排序的数组是一个排序数组或者是一个用分类机表示的数组吗？如果你相信类型地址对这些问题都进行了设计，那你就必须对这 个问题做出判断。

我认为那是个可笑的方法去思考编程。重要的不是事物间的祖先关系，而是它们可以为你做什么。

那是与我相关的细节。

从C++和Java转向Go的程序员不懂得使用类型编程的想法，尤其是继承、子类和与之相关的内容。也许关于类型我是个门外汉，但我还没发现一个模型能有如此的表现力。

我的一位老友Alain Fournier曾对我说，他把

学术工作的最低层次看作是分类学。你知道吗？类型分层恰恰就是分类学。你需要决定哪块进什么箱子，每个类型的父类，无论是 A继承于B还是B继承于A。一个可排序的数组是一个排序数组或者是一个用分类机表示的数组吗？如果你相信类型地址对这些问题都进行了设计，那你就必须对这 个问题做出判断。

我认为那是个可笑的方法去思考编程。重要的不是事物间的祖先关系，而是它们可以为你做什么。

我们应该使用一些方法，像花园浇水软管一样来连接程序——当需要用另一种方法处理数据时，犹如拧水管一样进入另一个分支。这也是IO的方式。

这也是Go的方式。Go接受了这个想法并发扬光大。它是一个构造和耦合的语言。

很明显的例子是它提供给我们构造成分的接口方式。成分是什么不重要，如果它实现了方法M，我可以把它放在这里。

另一个重要的例子是并发性如何构造那些独立执行的计算。

甚至有一个非寻常（非常简单）的类型构造形式：嵌入。

这些构造技术正是Go的风味特色，这是和C++或者Java程序的最大的区别。

我想提一下Go设计中的一个不相干的方面：Go是为了写大型程序、被大型团队编写并维护而设计的。

关于“大型编程”的想法，从某种角度看C++和Java占据着这个领域。我认为这仅仅是个历史偶然现象，或者是个业界偶然现象。但大部分人都觉得这和面向对象设计有某种关系。

我一点也不相信这些。大型软件当然需要方法论，但更加需要的是强依赖性的管理、干净的接口抽象和优秀的文档工具，而这些C++都不能很好的处理（尽管Java明显好些）。

我们现在还不知道，因为还没有足够的用Go写的软件，但我相信Go将证明是一个优秀的用于大型编程的语言。时间将证明一切。

现在是时候回答我们刚开始提出的令人惊讶的问题：

为什么Go，一种完全为C++是怎样用所设计的语言，又不吸引更多的C++程序员？

玩笑撇在一边，我认为其原因是Go和C++有着完全不同的哲学。

C++是可以让你在指尖拥有一切。我在C++11FAQ中发现了如下引用：

C++抽象范围可表示的很优雅，灵活，手工制作的专业代码和零成本相比是大大增加的。

这样的想法和Go的操作方式是不同的，零成本可不是Go的目标，但至少不是CPU零成本。Go 宣称极小化编程人员的付出是它们认为最重要的。

Go 语言不是包罗万象的。Go 并不自带很多函数，不会对每一个执行细节都有精确的控制。例如，你不会有 RAII。相反地，Go 语言有一个垃圾收集器（Garbage Collector，即GC），因此，不需要一个释放内存的函数。

Go 语言给你的是一套强大且易于理解、易于构建的对问题解决方案。它不是一种像你用其他语言一样的快而复杂、或者说思想上有所激发的语言，但它几乎肯定会更容易编写、更易于阅读、更容易理解、更容易维护，也许还更安全。

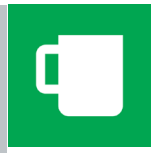
用另一种方式来说，当然过于简洁：

Python和Ruby程序员转向Go，因为他们不需要屈服于大量的表达形式，但可以提升性能，并能使用并发性。

C++程序员没有转向Go，因为他们为精确控制他们的编程领域付出了很多，不想屈服与它的任一方面。对他们而言，软件并不仅仅意味着搞定工作，它意味着使用某种确定的方法去做。

那么，问题是Go的成功将与他们的世界观冲突。

我们本应该从开始时就意识到这一点。那些因为C++11的新特性而激动不已的人是不会关心一个有更少特性的语言。即使，最终它能提供更多。■

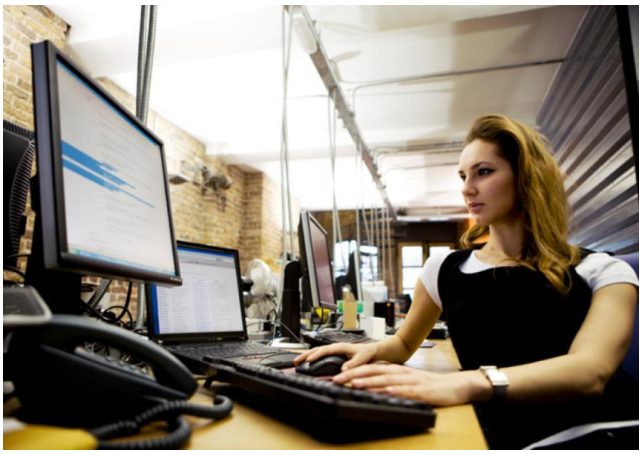


“你最喜欢的程序员漫画”精选--IE

编者按

当我们感觉自己很擅长一件事的时候，才会真正地去学习它，花费大量的时间和精力，全身心投入，直到非常精通为止。

为什么女程序员会这么少？



下面是译文全文：

当我们感觉自己很擅长一件事的时候，才会真正地去学习它，花费大量的时间和精力，全身心投入，直到非常精通为止。.

这种自信的感觉是一种强大的动力，只要我们想要做就一定能做到的信念称为自我效能（self-efficacy）。面对一项具体的工作，自我效能的来源有四种（按强度大小排序）：

- 亲身参与的成败经验
- 周围环境的影响(看见和我一样的人也在做)
- 社交关系的影响
- 身体状况的差异

为什么女程序员会这么少呢？因为很多女性感觉自己可能做不到，所以不愿去尝试，或者不再坚持。 这种自我效能感的来源解释了，为什么相对于男性而言，女性（总体上）不太可能会从事编程工作。

1. 亲身参与的成败经验：如果你尝试去做一件事，并且取得了成功，这就是自我效能感的最好来源。就我们这一代人来说，有很多男生会比女生更早地开始接触编程。

2. 周围环境的影响：如果妈妈能做到，我也能做到。

人们在选择职业的时候，往往会想象自己从事这份工作时的情形，这种想象 是建立在自己所认识人的基础上。如果无法想象自己从事这份工作的样子，那就压根儿不会考虑它。在我们的文化中，性别的差异很明显。我们常常都会顾虑和参考 身边跟我们差不多的人。男生可以想象自己是一名程序员，而女生环顾四周，去看不到和她一样参与编码，参加会议，做演讲，写博客，参与开源的女性。即使是女 性开发者：纵观职业层次，她们会认为自己从事管理，分析，QA，BI，或者DBA会更好，而不是去做系统管理员或者架构师。

3. 社交关系的影响：我的朋友们会同意吗？

这里，我们不是在讲编程文化，而是女性文化。我去妈咪晚间幼儿园（Kindergarten Moms

Night）时，会有人问我是做什么工作的，我回答“计算机编程”，通常对话就会结束了，我无法继续这样的谈话。在长大成人的过程中，如果你很享受和女性同伴们的相处，那么这种社交关系势必会影响自身的发展。

4. 身体状况的差异： 在编写程序时，你身体有没有感觉到不舒服我不知道，这对于男性和女性有差别吗?

女性在编程能力上和男性是实力相当的，但是（总体上）女性没有感觉到自己的能力。如果我们能够改变自我感受的能力，就能够改变程序员男多女少的比例。

第一条是最直接的途径，社区很多人都在努力为年轻人普及编程，尤其是女生，为他们鼓掌!

第二条也是我们可以改变的。增加现有女性开发者的曝光率，特别是那些高级别，高水平的女性。我们都期待看到周围那些进行决策，制定策略的女性。特别感谢想方设法吸引和招纳女性的企业和单位，你们实在功不可没!

第三条来自更大层面的社会文化，而不是程序员文化。 我做不到既是一位典范妈妈，又是一名参与社区工作的开发人员。这不是编程社区可以改变的。在我看来，第三条对女性来说是最棘手的麻烦事儿。

作为一个社区，第一条和第二条是我们力所能及的事，况且，它们还位居前两条。如果我们继续努力，就会达到一个临界值。一旦女性程序员比例达到 33%，那么第三条自然而然也就能实现了。在不受外界干扰的情况下，社会压力和缺乏典范使得越来越少的女性参与和坚持程序开发的工作。只有努力， 我们才能扭转这种可悲的局面。■

微软揭晓Visual Studio 2013售价和推出计划

Visual Studio 2012和Visual Studio 2013在SUK阵容方面略有不同，微软将为运行Visual Studio 2012 Professional的用户加入一个Visual Studio Professional 2013 Upgrade SKU。在促销期间（从2013年11月1日到2014年1月31日），有意购买VS Professional 2013 Upgrade SKU的用户可在微软网上商店（Microsoft Online Store）以99美元买到它。错过这段时间，其售价将上升到299美元。

当前的Visual Studio 2012售价包括:

- VisualStudio2012Professional: $499
- Visual Studio 2012 Professional with MSDN: $799
- Visual Studio 2012 Premium with MSDN: $2,569
- Visual Studio 2012 Ultimate with MSDN: $4,249

微软在今年9月推出Visual Studio 2013的候选发布版。根据微软VS 2013定价页面的信息，Visual Studio 2013将会在11月13号正式推出，但MSDN订阅用户将可以在10月18号访问Visual Studio 2013 RTM，那天也是Windows 8.1常规发售的日子。带MSDN订阅的Visual Studio 2013用户将可以在11月1号获得批量许可，而零售则还要再等几天。

Visual Studio 2013和Net 4.5.1将为C#, VB, JavaScript和C++开发者新增支持异步调试，不过需要Visual Studio 2013在Windows 8.1上运行才行，其他旧版本的Windows并不支持。当然，除此之外也还加入其他的新功能。最新版VS还将为那些使用XAML、HTML和JavaScript来开发Windows Store/Metro应用的开发者进行了一些改善。微软在今年6月下旬还推出了Visual Studio 2013首个公开预览版本。■

评论：“走出丛林”的马云和阿里巴巴

□ 新浪科技 /文

这次，马云和阿里巴巴挑战香港证交所的股票上市规则，战况绵延持久。我真心希望马云和阿里巴巴能赢，在香港顺利上市。

采用阿里巴巴自定义的“合伙人”制度在香港上市，符合阿里巴巴的利益，符合软银和雅虎两个大股东的利益，当然最符合马云本人的利益——间接控制合伙人、左右董事会席位，以较少的股份继续实现对公司的控制。但是别忘了，这么做也符合接下来那些希望在香港上市的创业者、特别是科技创业者的利益：一个新的制度被创造出来了——创始人不会因为上市股份被大量稀释而丧失对公司方向的控制权，从而被资本左右随波逐流。

如果你认为 Google 和 Facebook 在美国证券交易市场上通过“双层结构”的设置，让A级股票和B级股票拥有完全不同分量的投票权，并通过“表决权代理协议”的方式让佩奇、布林和扎克伯格们继续保证对公司控制权是正确决定的话，那么你没理由从一开始就认为：马云图谋在不限制股票减持的前提下继续控制公司方向的“制度创新”，是自私、破坏规则和罔顾股东利益的行为——再说了，目前阿里巴巴的两大股东——软银和雅虎，都表态支持阿里巴巴的这一举措。

唯一的问题是：上一轮上市之后的股价表现和以及 2011 年在支付宝 VIE 事件中的做法，让美国的资本市场对阿里巴巴和马云本人顾虑重重，以致按 Google 和 Facebook 开创的“先例”行使控制权并顺利在美上市的难度变得前所未有。在美国资本市场，马云和阿里巴巴的信用值的确大打折扣。但还是那句话，如果不考虑马云的个人算盘，他发明的“合伙人制度”在香港资本市场真正被接受的话，受益的将不仅是阿里巴巴本身。至少，我们可能会少听到那些公司创始人被资本意志驱逐的故事了——越来越多的例子证明，至少在互联网和科技行业，没有创始人掌舵的公司都死得快。

法律不应该成为制度创新的扼杀者，而是应为公司制度创新留下空间，由市场检验创新制度的生命力——这样的常识毋庸赘述。但阿里巴巴和马云挑战香港的证券监管制度，其难度之高，背后的波诡云谲与一波三折，加上外界的毁誉参半，都让这件事更具戏剧感。到现在我仍然乐观相信，阿里巴巴一定会在香港上市。美国和完全不靠谱的中国内地，都是虚晃一枪。桥段越多，事情越显得惊险。

于是我也不得不感叹，这么不着边际的疯狂挑战，在中国互联网界，只有马云敢这么玩。

看看马云每次都是挑战谁：上市事件挑战香港证券监管的体制；”阿里金融”挑战国有银行和传

统信贷、授信和金融产品；“菜鸟物流” 挑战的是传统的基础物流建设方式……这几件“大事”背后的脉络都很清晰：第一，“被挑战者”或是壁垒极其森严的传统行业，或是有行政垄断资源优势的山头势力，马云知道它们的底线在哪儿，不是打擦边球，而是小心翼翼地“越界”，然后倒逼对方把红线向里收缩；第二，为了挑战这些传统势力，马云往往也会借重传统势力的另外一方力量，比如以支付宝为主体的阿里金融(此次被剥离出上市范围)引入了中信资本、博裕资本和国家开发银行等背景雄厚的投资者；第三，这些挑战都将马云的野心和阿里巴巴的利益最大化，但同时也让阿里巴巴的用户、同行甚至敌人的利益最大化，比如挑战香港证券监管体系将有助于其它科技公司赴港上市，阿里金融将为用户提供更快捷和互联网化的服务，“菜鸟网络”不但让阿里巴巴电子商务的物流效率最大化，也让同行甚至对手们的物流因此受益。

当然也有例外：挑战 VIE 架构伤及整个中概股市场和国内绝大多数科技公司；挑战科学(没错我说的就是李一)伤及自身声誉；他似乎不久前还挑战过整个中国知识界的另一个共识，点到为止。

但你不能否认，这样的格局，在中国的互联网公司中只有阿里巴巴具备；中国的互联网大佬中，只有马云敢这么想，而且这么做了——哪怕有点忘乎所以。

如果让我一句话道出马云和马化腾的区别，那就是：在自身的利益扩张碰触到传统的既得利益和传统体系的时候，马云选择突破边界，继续颠覆；马化腾选择就此收手，妥协合作。

马云曾公开表露过他要改变银行业的心迹；而当腾讯申请银行牌照的时候，马化腾却赶忙出来澄清腾讯做支付“只为支持政府政策，与银行合作”。而更典型的例子是，类似 Whatsapp、LINE 和微信这样的应用势必会改变人们过去通过电信运营商网络收发短信和购买增值服务的方式，但腾讯最终选择让微信与运营商合作推出“沃卡”，以打消电信运营商的顾虑。

你经常会听到马云在公开场合大谈“理想主义”，听多了是有些矫情，但如果从这一件件事来看的话，他确实比更多中国互联网的“大佬”们更有理想。另外补充一句，马云和他的夫人不久前刚成为“生命科学突破奖”基金会的捐助人，该基金主要针对生命医疗特别是癌症领域的研究突破。在马云之前，该基金的捐助人只有 Google 联合创始人谢尔盖·布林(Sergey Brin)夫妇、Facebook 创始人马克·扎克伯格(Mark Zuckerberg)夫妇和俄罗斯著名投资人 Yuri Milner。

在用互联网改变中国人生活这件事上，阿里巴巴确实是做得最疯狂的一个。阿里巴巴和马云挑战的那些对手——那些监管机构、国有银行和骨干物流体系，是国内大多数互联网巨头和准巨头们从来不曾试图挑战的。而另一方面，阿里巴巴对其它巨头和准巨头们的“丛林游戏”似乎也并不感兴趣。尽管阿里巴巴比任何一个国内的互联网同行都擅长操作舆论，阿里巴巴也会推出“来往”这样试图与微信一争高下的产品，尽管阿里巴巴的投资与并购出手之快速决绝也让其它同行感到压力，但你会发现这些都是因为它太在意自己会成为什么。阿里巴巴很少为了狙击某个竞争对手，而故意投资竞争对手的对手甚至结盟，更不会去参与那些几家巨头准巨头围殴另一个巨头准巨头的“丛林之战”。

只有走出丛林，游戏才更精彩。■

为什么有这么多Python?

□ 开源中国 / 编译

Python是出类拔萃的

然而，这是一句非常模棱两可的话。这里的”Python”到底指的是什么？ 是Python的抽象接口吗？是Python的通用实现CPython吗（不要把CPython跟Cython搞混了）？亦或者指的完全是其他的东西呢？ 可能我另外指的是Jython，或者IronPython，或者是PyPy。也或者转而谈论的又是RPython或者RubyPython（这两者是完全 不同的东西）。

上面提到的那些技术经常被提起和引用， 它们的使用目的和场景是完全不一样的（至少，它们的操作方式是完全不一样的）

自从我使用Python工作以来，我已经用过了各种各样的.*ython工具了。但是直到最近我才花时间去理解到底它们是干嘛的，它们是怎样工作的，为什么它们是不可或缺的。

在这篇文章里面，我会介绍各种Python的实现，最后以对PyPy的介绍结尾， 因为我个人认为它是Python的未来。

所有的都从理解什么是”Python”开始。

如果你对机器码，虚拟机之类的很熟了，你可以跳过开头，直接从 “即时编译: PyPy和它的未来” 这部分开始看起。

Python是解释型的还是编译型的?

这是个Python新人都会迷惑的问题。

首先需要明了的是Python只是一个接口。有一个关于Python应该做什么以及怎么做的具体说明(就像其他任何接口一样)，并且对应的有很多具体的实现（也像其他接口一样）。

其次需要知道的是“解释型”和“编译型”是具体实现的特性，而不是接口的特性。

所以，这个问题本身就没有组织好。

Python是解释型还是编译型的？这个问题真的没有组织好。

对使用最广泛的实现（CPython:用C实现的，通常简单的说成Python，若你不知道我所说的这些，那很肯能你在使用的就是CPython）而言，这个问题的答案是：解释型，但带有一些编译型特征。CPython把Python源码编译*成字节码，之后再解释这些字节码，执行之。

*注意：这个编译不是通常意义上的编译。通常我们说的编译，是指把高级语言代码转换成机器码。但这里的编译实际上是另一种意义上的编译。(译者，这句话不是很懂，原文是it is a ‘compilation’ of sorts，不知作何解，求教各位读者。)

再详细看下上面的答案吧，这有助于我们理解本文中后面会讲到的几个概念。

字节码 vs. 机器码

了解字节码和机器码（或者native code）的区别是很重要的，最好的办法或许是看看例子：

C代码被编译成机器码，将在处理器上直接执行。每一条指令控制CPU工作。

Java代码被编译成字节码，将在Java虚拟机(JVM)这个抽象的计算机上执行。每一条指令由JVM处理，JVM同计算机本身之间交互。

机器码随机器的变化而变化，但字节码在所有的机器上都是一样的。有人可能会认为机器码是对特定环境优化了的。

回到CPython，工具链的执行过程如下：

CPython编译你的Python源代码，生成字节码。

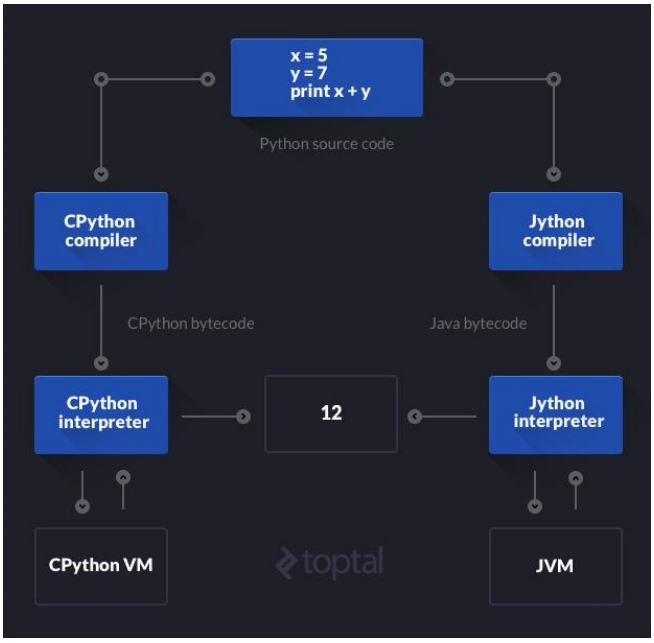
字节码随后在CPython虚拟机上执行。

初学者常常因为看到.pyc文件而假设Python是编译型的。这也有一些合理性：.pyc文件正式之后要解释的字节码文件。所以，你若之前运行过你的Python代码，生成了.pyc文件，再次运行时就要快得多，因为不需要再次编译生成字节码了。

可选的虚拟机：Jython，IronPython等

正如我之前所述，Python有很多实现。前面也提到，CPython是最通用的。这是一个用C实现的，被认为是”默认“的实现。

但其他的呢？其中最显赫的之一就是Jython，一个用Java实现的采用了JVM的实现。CPython生成在CPython虚拟机上运行的字节码，而Jython生成在JVM上运行的java字节码(这同编译Java程序生成java字节码的过程是一样的)。



”为啥你要用其他的实现？”，你可能会如此发问。好吧，对开发者而言，不同的实现对不同的技术难题的支持程度不一样。

CPython中很容易为你的Python代码写C扩展，因为最终都是由C解释器执行的。另一方面，Jython则使得和其他java程序共同工作很容易：无需其他工作，你就可导入任何Java类，在你的Jython程序中使用其他Java类。（题外话，若你没有认真思考，这一段会很难。此时我们已经在 讨论把不同语言的代码混在一起，并编译成同一程序。（Rostin 提出混合Fortran和C代码编程已经有一段时间了。所以，这并不新鲜，但仍然很酷。））

下面是一个例子，一段合法的Jython代码：

```
1. [Java HotSpot(TM) 64-Bit Server VM (Apple
   Inc.)] on java1.6.0_51
2. >>> from java.util import HashSet
3. >>> s = HashSet(5)
4. >>> s.add(“Foo”)
```

5. >>> s.add(“Bar”)

6. >>> s

7. [Foo, Bar]

IronPython是另一很流行的Python 实现，完全用C#实现，针对.NET平台。她运行在可以叫做.NET虚拟机的平台上，这是微软的 Common Language Runtime (CLR)，同JVM相对应。

你可能会说，Jython:Java::IronPython:C#。它们各自运行在相同的虚拟机上，你能从你的IronPython中导入C#的类，从你写的Jython代码中带入Java类，等等

你完全可以不用任何非CPython的实现就能完成你手上的任何工作。但是使用这些技术也是有很多的好处的，大部分取决于你现在所使用的技术栈。 你使用了很多基于JVM的语言？Jython就是为你准备的。使用的都是.NET世界的语言？那么你应该试试IronPython了（或许你已经在用了）

Implementation	Virtual Machine	Ex) Compatible Language
CPython	CPython VM	C
Jython	JVM	Java
IronPython	CLR	C#
Brython	Javascript engine (e.g., V8)	JavaScript
RubyPython	Ruby VM	Ruby

toptal

顺便说一下（尽管这不是使用不同的实现的理由），注意Python的各种实现在对待你的Python源码的时候所做的处理方式是完全不一样的。然后这些差异是很小的，由于这些实现都在不停的发展改进中，随着时间的推移，这些差异会慢慢融合和兼容。比如，IronPython默认情况下使用Unicode字符串，但是在2.x版本的CPython中默认是ASCII字符串（如果使用了非ASCII字符串，会抛出一个UnicodeEncodeError错误），但是在3.x版本里面CPythong已经默认支持Unicode字符串了。

即时编译: PyPy和它的未来

我们已经有了一个使用C写的Python实现，一个用Java写的，一个用C#写的。接下来就是：用Python写的Python实现（有心人可能会注意这句话有点问题，是个死循环，^_^）

接下来我们看下什么地方容易搞混淆。首先，我们讨论下即时编译器JIT

JIT: 为什么会有这个？它的原理是什么？

大家都知道本地机器码的速度比字节码的速度快很多。那么，如果我们能将一些字节码直接编译成本地机器码再去运行它会怎样呢？我们必须花费一些代价（比如时间）在编译字节码到本地机器码上，如果最终的运行时间更快，那么这个代价就是值得的。这就是JIT编译器的动机，一种混合了解释器和编译器好处的技术。简单来讲，JIT就是想通过编译技术提升脚本解释器系统的速度。

例如, 被JIT（及时编译）采用的通用方法:

1.标识被经常执行的字节码。

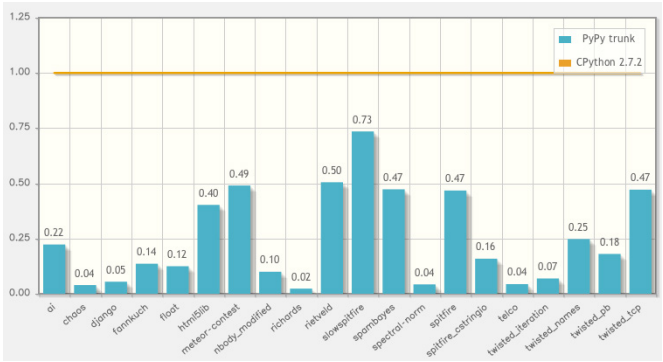
2. 把其编译成本地的机器码。

3. 缓存该结果。

4. 当同样的的字节码再次被执行的时候，会取预编译的机器码，得到好处（例如速度提升）。

这是关于PyPy的用处: 把JIT代入Python语言（参看前面成果的附录).当然也有其他目的: PyPy 目标是成为一个跨平台，轻内存，支持stackless（译注：stackless为python提供微线程扩展，具有并发特性）。 但是及时编译才是它真正的卖点。

基于一系列时间测试的平均，据说性能上能提高6.27倍. 停一下，看看下面这个由PyPy Speed Center提供的图表:



PyPy is Hard to Understand

PyPy具有巨大的潜力，在这一点上，它与CPython高度兼容所以它能运行Flask，Django等等）。

但关于PyPy有许多困惑（例如，荒谬的建议创造一种PyPyPy…语言). 按我的观点，那主要是因为PyPy实际上是两种东西:

一种用RPython（非Python（我之前撒谎了））编写的Python解释器。 RPython是Python的子集，具有静态类型。在Python里，最难严格推论类型（为什么这么困难，考虑下下面的事实:

x = random.choice([1, “foo”])

1.将是合法的Python代码 (归功于 Ademan). x的类型是什么？我们怎么推出变量的类型，当类型还没有被严格实施?)通过RPython,你牺牲了一些灵活性, 但使得内存管理和优化大大的容易。

2.一个编译RPython代码为了各种目标和加入及时编译的编译器。 默认平台是C，也就是从RPython到C编译器，但你也可以瞄准JVM或者其他。

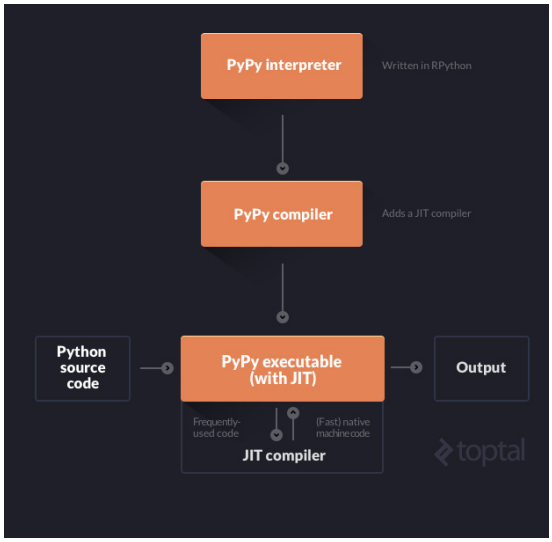
只为清晰，我将引用这些PyPy（1）和PyPy（2）。

为什么你在同一层面下同时需要这两者？ 你可以这样想一下：PyPy(1)是一个用RPython写的解释器，因此它能加载用户的Python代码并将它编译成字节码。但是这个用RPython写的解释器本身要能运行，就必须要被另外一个Python实现去解释，对不?

我们可以直接用CPython去运行这个解释器。但是这个还不够快

取而代之，我们使用了PyPy(2)（参考RPython的工具链）去编译这个PyPy的解释器，生成其他平台（比如C，JVM或CLI）代码在我们的机器上运行，并且还加入了JIT特性。这个很神奇：PyPy动态的将JIT加入一个解释器，生成它自己编译器！（这就是核心原理：我们在编译一个解释器，并同时加入了另外一个单独的编译器到里面去）。

最终结果就是一个融合了JIT优化特性的单独的可执行文件，用来解释执行我们的Python源代码。这就是我们之前想要达到的效果。这么讲可能比较拗口，下面这张图可能会解释的比较清楚点:



再次重申下，PyPy真正可贵之处在于我们可以利用RPython实现各种不同的Python解释器，不用去关心JIT（除了一些小的提示外）。PyPy到时候会利用RPython工具链/PyPy(2)为我们自动实现JIT。

事实上，我们还可以更抽象一点，我们理论上可以写一个适用于任何语言的解释器，然后将它扔给PyPy，最后获得那种语言的JIT。原因是PyPy仅仅关心的是优化解释器，而不会去关心这个解释器到底解释的是什么语言。

理论上你自己可以写一个适用于任何语言的解释器，然后将这个解释器传给PyPy，最后你得到这个语言的一个JIT。一个简单的题外话，我这里想提一下，JIT本事是相当棒的。它使用了一种叫做跟踪的技术，按照下面的步骤执行：

1.执行解释器并解释执行所有代码（还没有加入JIT特性）

2.对被解释过的代码做一些记录

3.确认你已经执行过的操作

4.将确认过的这些代码编译成本地机器码

想获取更多信息，可以参考这篇文章，易于理解，并且非常有趣

最后收尾：我们使用PyPy的RPython-to-C（或者其他目标平台）编译器去编译PyPy的基于RPython实现的解释器。

结尾

为什么它如此的伟大？为什么这个疯狂的想法值得我们去追求？我想Alex Gaynor已经在他的博客上面做了很好的解释了：“[PyPy就是未来] 因为[它]提供了更快的速度，更大的灵活性，并且对于Python的成长也提供了一个更好的平台”

总之：

・它很快，因为它将源代码编译成了本地机器码（使用了JIT）

・它很灵活，因为除了极少数的额外工作需要做外，它就能将JIT加入你的解释器中

・它还是很灵活，因为你能使用RPython实现你的解释器，这个比其他的（比如C语言）更易扩展。事实上，它是如此的简单，这里有一篇教程教你如何实现你自己的解释器。

附录: 其他一些你可能已经听过的名字

・Python 3000 (Py3k): Python 3.0的一个别名，2008年释出的一个主要版本，但是它并不向后兼容.。Py3k团队预测这个版本被完全采用可能需要5年时间.。现在绝大多数（注意：这个是江湖传闻）Python开发者继续在使用2.x版本，不过现在人们越来越多的对Py3k开始关心了。

・Cython: 一个Python的超集，能够调用C语言的函数 ・ 目标: 允许你为你的Python代码写C扩展

・ 允许你为你的Python代码加入静态类型，运行编译并达到接近C语言的性能。

・ 这个跟PyPy比较类似，但是不是一样的。使用这个的时候，在提交给编译器之前必须用户代码里面写好这些特殊代码。如果使用PyPy的话，你写的还是普通形式的Python代码，编译器会帮你处理一切优化的工作。

・Numba: 将JIT加入到被注解的Python代码

中，简单来讲就是，你给它一些提示，它就会优化加速你这段代码。Numba是Anaconda发行版（一系列数据分析和管理的软件包）的一部分。

· IPython: 跟我们讨论过的其他版本完全不一样。这是一个Python的计算环境。为一些GUI工具集和浏览器体验等提供支持。

· Psyco: 一个Python的扩展模块，也是早先的一种Python JIT的成果。 然而，它已经被标注为“停止维护和死亡”了。事实上，Psyco的首席开发者Armin Rigo现在在为PyPy工作。

语言绑定

· RubyPython: Ruby和Python虚拟机的一座桥梁。允许你在你的Ruby代码中嵌入Python代码。你定义Python的起始位置，然后RubyPython负责在不同VM直接传递整理数据。

· PyObjc: Python和Objective-C语言直接的桥梁。实际上，这意味着你能在你的Python代码中使用Objective-C的库（包括创建一个OS X应用程序所需要的一切），反过来在Objective-C里面也可以使用Python的模块。这样的话，CPython用C语言来实现就很方便了，因为C语言是Objective-C的一个子集。

· PyQt: 同PyObjc帮你绑定OS X GUI组件类似，PyQt帮你绑定Qt应用程序框架，让你可以创建丰富的图形界面，访问关系数据库等等。另外的一个旨在帮你简化从Python到另外的框架的工具。

JavaScript 框架

· pyjs (Pyjamas): Python中一个创建web和桌面应用程序的框架。包含一个Python-to-JavaScript的编译器和其他一些工具。

· Brython: 一个使用JavaScript语言写的Python虚拟机，可以让Py2k 代码在浏览器中执行。■

链 接

Java 8发出提示 可能将不再支持Windows XP

今天安装Java JDK 8 108版时安装程序弹出提示，大意是 Java 8 需要更新版本的Windows。你可以继续安装但是为了Java能正常的运行，我们建议你升级你电脑的操作系统。

值得注意的是，上一个JDK8的预览版106版还没有提示，今天发布的108版开始提示要去升级系统，这意味着等明年年初Java8正式发布时很可能不再支持XP系统。

开 心

以我个人的经验，只有在做一些相对重复性的工作，或没有压力的工作时，我才喜欢播放一些背景轻音乐。我并不认为这…

音乐对编程的影响

在20世纪60年代期间，研究人员在康耐尔大学进行了一系列有关在音乐背景下进行工作的效果测试。他们对一组微机科学专业的学生进行了调查，把学生 分成了两 组，一组喜欢边听音乐边工作，另一组不喜欢这样做。然后把他们每组中的一半人带进一个安静的教室，把另外一半人带进一个配备了耳机和音乐选择功能的教室， 给两个教室中参与调查的人一个同样的Fortran编程问题，让他们根据说明加以解答。结果是，两个教室中参与调查的人以同样的速度和同样的编程准确度在 解答问题，这一点没有人会感到惊讶。正如任何边听流行音乐边做算术家庭作业的小孩那样，算术需要的、与逻辑有关的那部分大脑没有受音乐的干扰——而有另外 一部分大脑在听音乐。

但是康耐尔实验包含了一张隐蔽的百搭牌。题目说明要求通过一系列的操纵输入数据流中的号码来形成输出数据流。例如，参 与调查的人必须移动每个号码左边的两位数字然后除以一百等等。虽然题目说明并没有直说，但是所有运算的最终效果是每个输出号码必须等于它的输入号码。有些 人意识到了这一点，但是有些人没有意识到这一点。那些意识到了这一点的人，绝大多数来自那个安静的教室。

专业员工每天做的事情中，许多是由 左脑的顺序处理中心完成的。音乐不会特别干扰工作，因为是大脑的整个右边在消化音乐。但不是所有的工作都由左脑完成。可能有让你说“啊！”的突破会引导你 到达一个可以节约数月或数年工作的创造性思路。创造性的飞跃包括在右脑的功能中，如果右脑忙于听背景音乐台的10001弦乐，那么就有失去创

造性飞跃的可 能性。

环境造成的创造性方面的惩罚是潜在的。因为创造性是一种损失时我们经常注意不到的东西。创造性减少的影响是一个很长的日积月累的过程。公司越没有生产力，人们就越会没有激情的火花，只会机械地工作，最优秀的人便会离开。

——《人件》

以我个人的经验，只有在做一些相对重复性的工作，或没有压力的工作时，我才喜欢播放一些背景轻音乐。我并不认为这能让我提高工作效率，只是会让我更开心些。而深度思考时需要安静，需要排除脑子里任何的杂音，固化那稍纵即逝、易碎的灵感。

对于很多人来说，相对于播放轻音乐，安静可能是对他们的编程来说是最好的选择。而不幸的是，程序员通常不是在这两者之间做选择，他们要选择的是要音 乐还是办 公室里使人无法集中注意力的噪音(周围人的交谈，人们移动的声音，等等)。在这种情况下，任何能阻挡这些背景噪音的音乐都是恩赐。带着耳麦，这就是向访客 传递了信号，我很忙，没有重要的事情不要打搅我。

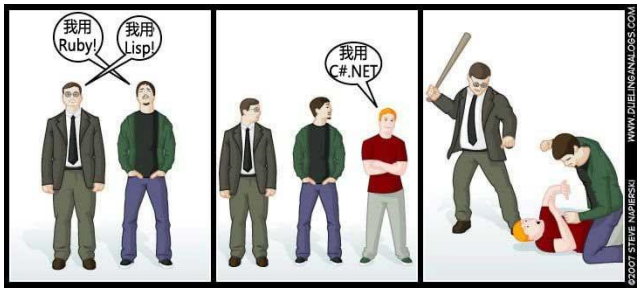
《人件》是 本很有名的书，也是本让人非常收益的书。一位读者碰巧在一次会议上碰到了这本书的作者之一，于是向他表达式如何的喜欢书中的一个故事(指荷兰东印度公司的 故事，它曾经是世界上最大的公司，如今仍存在，但只有50个做文书工作的员工)，并问他是从哪里找到这个故事的。“我们讲了这个故事吗？”他反问。当然， 绝对有，这位读者说。“哦”，他回答到。“我们通常只在演讲中杜撰一些故事，书中一般不用。”

你也许会感到吃惊，这么一本有名的书中竟然采用杜撰的故事吗？有兴趣的朋友可以打开这本书，翻到书末“注释”部分，其中第十二章是这样写的：

[1] 康耐尔实验从未公开发表，因为证据看来像传闻，除了我们提供的证据之外。

也就是说，上面的关于音乐对集中精力的影响的康耐尔实验是传闻。■

译文链接：http://www.aqee.net/the-effects-of-working-with-music/



做为技术人员为什么要写博客

■ 编者按

本文只代表个人见解，不代表任立场，如果您认为我的想法是错的那很正常，因为这是我的想法，如果您觉得您的想法和我一样，那我们就是传说中的 “激友”（对生活冲满激情的朋友）。进入正题

本文只代表个人见解，不代表任立场，如果您认为我的想法是错的那很正常，因为这是我的想法，如果您觉得您的想法和我一样，那我们就是传说中的 “激友”（对生活冲满激情的朋友）。进入正题

一、我心中的博客

我所以指的写博客，不单只是写一篇文章出来这一结果。而应该是写的这一过程，写过技术文章的朋友应该跟我一样有这么一个过程。

1 自己了解学习，文章所涉及到的知识点，及知识点衍生出来的知识点。

2 对学习的知识点进行验证，以确保理论值与实践值保持一致。

3 构思文章的大纲，哪些部分需要重点写，需要配合实例代码，图片等信息。

4 动手写，写完后再次检查校正并排版，然后发表。

5 针对网友的评论中提出的问题进行回复。

我写文章一般都会经历以上5上步，最终以上5步融合成一个结果那就是”一篇文章” 这一过程也是我心中对的”写博客”一词的诠释。

二、为什么要写博客

2.1 为自己

写博客对自己的提升是很大的，可能写一篇体现不出来，但是只要你坚持写效果就很明显，好处人个认为有以下几点：

2.1.1 强化知识点：在写一篇文章前，你必定是要把以文章中心为主的知识点及衍生的知识点都详细了解一篇， 在这一过程中必须会涉及到自己以前所了解过的知识，人的记忆是存在记忆曲线的需要不断的重复记忆才能长久的记住某一事物，而每写一篇文章时都会查阅资料， 在这一过程中必然会遇到以前记住了而现在渐渐淡忘的知识点，当你再次看到时瞬间就会回想起，此时以前的知识点就得到了强化。

2.1.2 提升学习能力：同一样的人，了解同一知识点，用不同的方法，产 生的结果必然会不一样，找到最佳的学习方法，这也是一种能力，这种能力是经过多次实践探索之后总结出来的，以前我每次需了解某一种技术时都会先百度看各种 搜索结果，发现没有想要的之后，再Google因为Google的结果与百度的会有所不同，Google结果中国外的文章相对会多一点，而偶然点了一个链 接进入了博客园，发现就是自己想要的东西，而且把概念，代码，及经验都写上去了，看完之后对我帮助很大。

渐渐的我便开始采这种方法了解新知识概念性的直接看百度百科，实质性的直接 上博客园的 找找看，群里的朋友还推荐了一种方法，比如我要学MVC园子里很多人都写了 关于MVC的一系列文章，把

那一系列的文章都看一遍，对于MVC就基本有了了解了，这便是学习能力的提升,对于某种技术用最短的时间做到了比较全面的了解

2.1.3 提升文字组织能力：这个就不用说了，写博客，既然是写，就必然会有大量的文字，而如何组织文字表达出自己想表达的意思，是长期练习的，而写博客正好帮助你提高了你的文字组织能力，

2.1.4 提升逻辑思维能力：不用说，技术性的东西从来就没的单独存在 的，都一层层技术相结合，那在了解某种技术时，自己的思维也是要顺着这种关系逐渐深入的，比如MVC，你不能只知道 M是什么V是什么 C是什么就行了吧，你得知道 M V C 这三者关系是怎样的，又是怎样交互，而你了解之后再把它写出来时，需要清晰逻辑。

2.2 为他人

2.2.1 有意的： 园子里有很多人都写过关于MVC 框架 WCF 等等系列文章，目的就在于帮助新人快速上手，这个我深有体会，当初我开始学习MVC时就是看的T2噬菌体的MVC系列文章，整篇看完后再配合自己动手对于MVC就有了基本的了解了，在次感谢园子里无私献的大牛们。而以上行为就是有意的帮助

2.2.2 无意的：很多时候在开发项目的过程中，遇到了技术问题，花了时间解决后，有人会写博客记录，并附上解决方法旨在当再次遇到问题时直接看下文章就知道如何处理了，

而碰巧的是，这种问题不止他一个人遇到了。很多人在开发时也遇到了这个问题，在网上找答案时，就找到了这篇文章，并根据文章提供的解决方法，顺利的解决了问题，这种帮助就是无意的帮助

三、一定要写博客吗

答案肯定是否定的，中国几百万的程序员，如果都写博客，那程序员的春天就来了，但是事实并非如此。而我所讲的 写博客 的产物并不只是一篇文章，更多的是，对自己能力的提升，自己对知识点的总结，而发表在博客上只是为了公开，还有很多人喜欢记录在云笔记里面。还有工作很 忙，没有空闲时间写出来，因为写技术性的文章，花的时间是很长的。写过的朋友都知道。

四、博客会给你带来哪些收获

古人云：一份耕耘，一份收获

以下这些是帮助他人而得到的一些认可，并非主观上去追求的

MVP：微软每年都颁发MVP给那些经常与其他专业人士分享知识和专业技能，受人尊敬、信任，而且平易近人的专家。而这个称号则是对你写的博客质量的肯定

知名度:文章写得好的人，技术水平肯定也很好，知道的人多了，知名度就有了，比如园子里排名前10的大家都知道，都看过他们的文章。

尊敬:对于技术界的大神，都是受人敬仰的，在园子里或者工作中也是一样的，在心里对大神们都是默默的佩服！至少我是这样啦，哈哈~也是我学习的榜样！

五、总结

我所认的写博客是对自己所了解知识的强化，分享，自身能力的提升。当然写博客只是一种方法而以，只要能达到提升自我的效果什么方法都可以的。

之前有看过一篇文章：《即便没有读者，你也要写博客》，其中也讲解了很多写博客的好处。

而我写博客是希望，能提升自己的综合能力，并把自己的知识与经验分享给大家，如果有幸我的分享帮助了一些人，那将使我更加欣慰。

另外国庆节就要来了，祝奋斗在一线的程序员小伙伴们，放假快乐，放下工作好好出去玩几天吧！

最后附上一句名言：有些事情你现在不去做，可能以后都不会有机会了！

注：我代表不了大家，所以以上观点只代表我个人。■

技巧

图形加速可令Java提速10倍

在今天举行的 Oracle JavaOne大会上，IBM Java CTO John Duimovich 说，GPU（图形处理单元）加速器内置了可观的非图形处理能力，因为 GPU 的并行设计可以让许多子处理器同时运行。

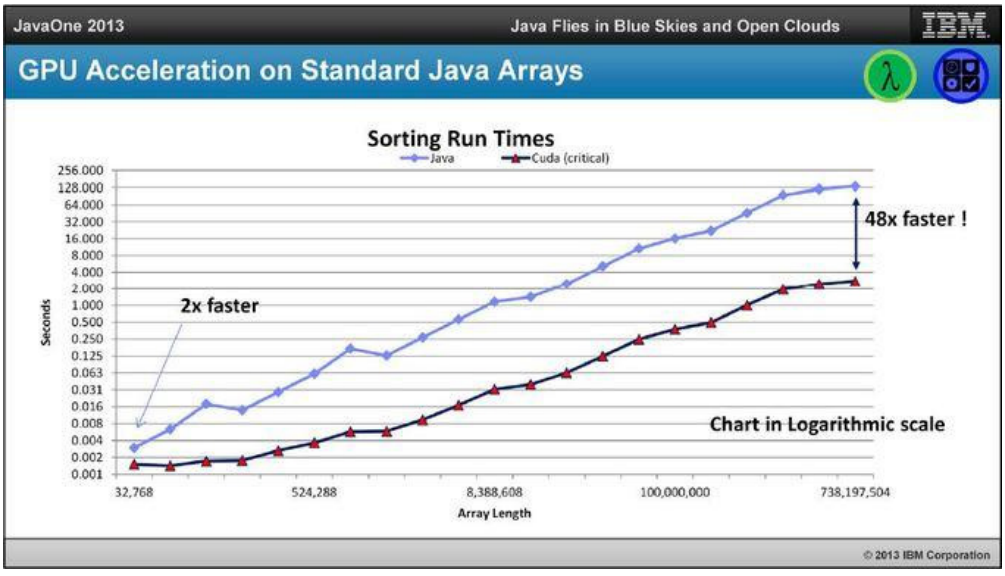
Nvidia 负责加速计算的总经理 Sumit Gupta 在一篇文章中称 Java 和 GPU 联手将会为加速 web 性能打开机遇之门。

数百万的开发者用Java语言进行Web 2.0开发、大数据分析及科学计算。同时，由于易于编程、模块化及对多平台的支持，Java也被用于大规模分布式的框架中，如 Apache，Hadoop 等。

一些现成的 GPU 库是基于 Nvidia 的 CUDA 计算环境开发的。开发者利用这些库可以将程序的性能提升2倍到48倍不等。下图是 Duimovich 举的一个例子：

Duimovich 说 IBM 将会针对基于服务器的 GPU 启用 IBM 运行时，并探索对基于现有 API 的一般负荷进行加速的可能性。

Gupta 则说，此举可令数百万的 Java 开发者利用 GPU 加速器对范围很广的应用进行加速，从而令应用性能大幅提高。此外，这类加速还会催生出一类必须依赖 GPU 的新型 Java 企业应用，包括高性能分布式的欺诈检测及金融分析，高通量视频及图像分析以及现代科学计算等。■



大型网站负载均衡架构

编者按

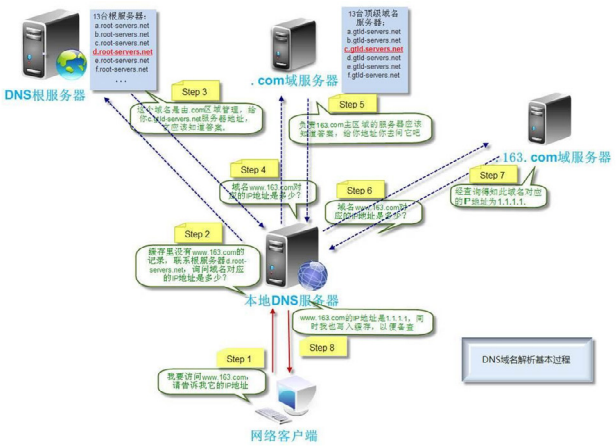
负载均衡（Load Balancing） 负载均衡建立在现有网络结构之上，它提供了一种廉价有效透明的方法扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。

负载均衡（Load Balancing） 负载均衡建立在现有网络结构之上，它提供了一种廉价有效透明的方法扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。

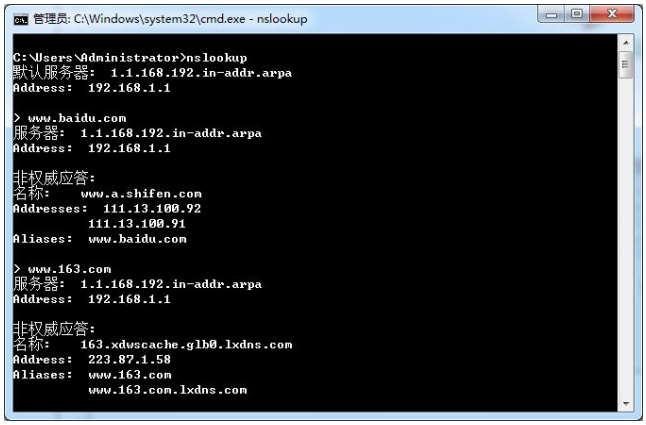
大型网站负载均衡的利器

- 全局负载均衡系统（GSLB）
- 内容缓存系统（CDN）
- 服务器负载均衡系统（SLB）

DNS域名解析的基本过程



最初的负载均衡解决方案（DNS轮询）



优点

基本上无成本，因为往往域名注册商的这种解析都是免费的；

部署方便，除了网络拓扑的简单扩增，新增的Web服务器只要增加一个公网IP即可

缺点

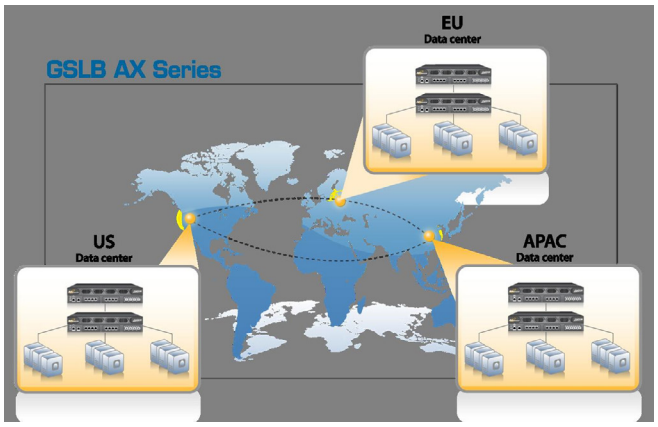
健康检查，如果某台服务器宕机，DNS服务器是无法知晓的，仍旧会将访问分配到此服务器。修改DNS记录全部生效起码要3-4小时，甚至更久；

分配不均，如果几台Web服务器之间的配置不同，能够承受的压力也就不同，但是DNS解析分配的访问却是均匀分配的。用户群的分配不均衡导致DNS解析的不均衡。

会话保持，如果是需要身份验证的网站，在不修改软件构架的情况下，这点是比较致命的，因为DNS解析无法将验证用户的访问持久分配到同一服

务器。虽然有一定的本地DNS缓存，但是很难保证在用户访问期间，本地DNS不过期，而重新查询服务器并指向新的服务器，那么原服务器保存的用户信息是无法被带到新服务器的，而且可能要求被重新认证身份，来回切换时间长了各台服务器都保存有用户不同的信息，对服务器资源也是一种浪费。

全局负载均衡系统（GSLB）



优势

- 数据中心冗余备份
- 多站点流量优化
- 确保用户体验

全局负载均衡系统（GSLB）的原理

DNS检查工具网上有很多，感兴趣的可以搜索一下。

内容缓存系统（CDN）

- 内容缓存系统（CDN）之静态加速
- 内容缓存系统（CDN）之动态加速

动态加速的特点

- 智能路由
- 传输控制协议（TCP）优化
- HTTP预载

服务器负载均衡系统

应用背景

- 访问流量快速增长
- 业务量不断提高

用户需求

- 希望获得7×24的不间断可用性及较快的系统反应时间

负载均衡必须满足性能、扩展、可靠性

服务器负载均衡系统三种接入方式

部署方式	特点	优点	缺点
串联路由模式	比较常见的部署方式	• 负载均衡设备将服务器有效隔离，安全考虑上最好 • 服务器网关指向负载均衡设备，功能实现更简单，有利于最大化负载均衡性能 • 服务器可以直接接收到真实访问源客户IP地址	• 对现有拓扑结构变动较大 • 需要考虑内网服务器是否有对外访问需求，必要时需要设置静态NAT转换
单臂模式	最常见的部署方式	• 部署方便，对现有拓扑结构变动小 • 和应用无关的流量不会通过负载均衡设备 • 内部应用无影响，外部应用通常需要前端防火墙做NAT映射到应用VIP	• 服务器不能直接接收访问客户源地址，需要对应用做修改后才可以通过其他方式获得真实访问地址
DSR	服务器回程报文不通过负载均衡设备，直接返回给客户端； 延迟短，适合流媒体等对延时要求较高应用	• 性能高，可处理吞吐量高 • 服务器可以直接接收到真实访问源客户IP地址	• 只能做4层的负载均衡，基于7层的服务无法实现优化(例如压缩等)无法使用 • 需要在服务器上配置loopback地址

服务器负载均衡系统的常见调度算法

- 轮询（Round Robin）
- 加权轮询（Weighted Round Robin）
- 最少连接（Least Connections）
- 加权最少连接（Weighted Least Connections）

健康性检查

健康性检查算法的目的：通过某种探针机制，检查服务器群中真实服务器的健康情况，避免把客户端的请求分发给出现故障的服务器，以提高业务

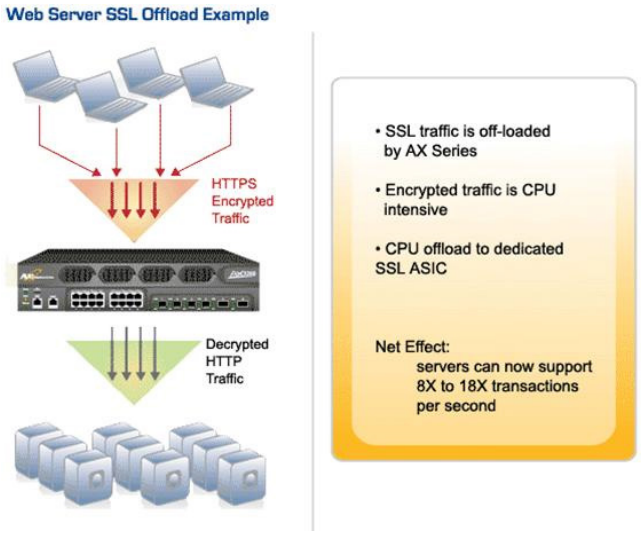
的HA能力。

目前常用的健康性检查算法：

- Ping（ICMP）
- TCP
- HTTP
- FTP

系统加速

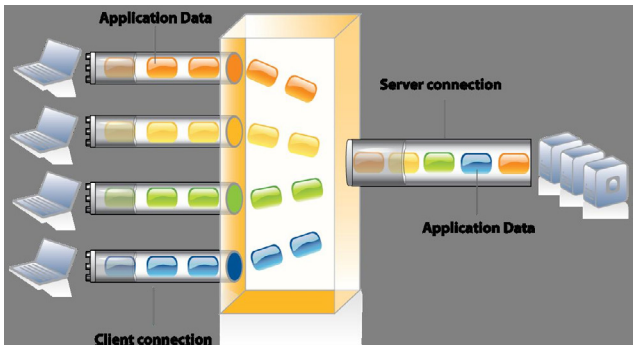
优化功能-SSL加速



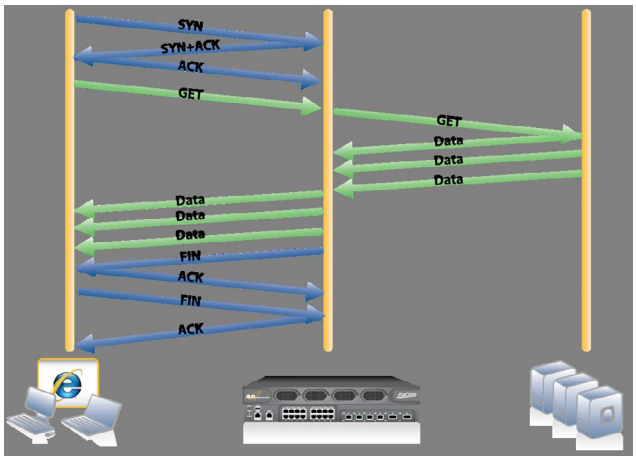
优化功能-HTTP压缩

HTTP压缩是在Web服务器和浏览器间传输压缩文本内容的方法。F5 HTTP压缩技术通过具有智能压缩能力的 BIG-IP 系统可缩短应用交付时间并优化带宽。HTTP压缩采用通用的压缩算法压缩HTML、JavaScript或CSS文件。压缩的最大好处就是降低了网络传输的数据量，从而提高客户端浏览器的访问速度。

优化功能-连接复用



优化功能-TCP缓存



会话保持

会话保持-客户端源IP会话保持

源IP地址会话保持就是将同一个源IP地址的连接或者请求认为是同一个用户，根据会话保持策略，在会话保持有效期内，将这些发自同一个源IP地址的连接/请求都转发到同一台服务器。

会话保持-Cookie会话保持

当采用基于源地址的会话保持无法做到负载均分时，例如客户端发起连接请求的源IP地址相对固定，发生此类问题通常可采用基于应用层的会话保持方式，Cookie通常是存在于HTTP头中，现如今基于HTTP的应用被广泛使用，因此基于Cookie的会话保持越来越多的出现在服务器负载均衡解决方案中。

局限性：

对于非HTTP协议，或者客户端禁用Cookie，无效。

会话保持-URL哈希（Hash）会话保持

哈希会话保持的一个基本概念就是按照某个Hash因子，根据此因子以及后台存在多少台服务器计算得到的结果来选择将请求分配到那台服务器。哈希会话保持的特点是在后台服务器的健康状态不发生改变的时候，每个特定的Hash因子被分配到的服务器是固定的。其最大的优势是哈希会话保持可以没有会话保持表，而仅仅是根据计算的结果来确定被分配到那台服务器，尤其在一些会话保持表查询的开销已经远远大于Hash计算开销的情况下，采用Hash会话保持可以提高系统的处理能力和响应速度。

URL哈希会话保持通常针对后台采用Cache服务器的应用场景，针对URL进行Hash计算，将同一个URL的请求分配到同一台Cache服务器，这样，对后台的Cache服务器群来说，每台Cache服务器上存放的内容都是不一样的，提高Cache服务器的利用率。

故障案例分析

Q&A案例分析（1）-循环跳转

故障现象：

Web服务端对用户访问的URL进行判断，对于非https的请求，重定向到http站点，结果导致用户一直302跳转。

原因分析：

采用了负载均衡SSL加速功能，在服务端看到所有的用户请求都来自于http。

解决方案：

全站启用SSL加速。

Q&A案例分析（2）-用户Session丢失

故障现象：

用户在http站点上提交数据到同域名的https站点，web程序抛出session丢失的异常，用户提交数据失败。

原因分析：

http和https在负载均衡设备上被认为是2个独立的服务，产生2个独立的TCP链接，会命中不同的真实服务器，导致session丢失。

解决方案：

在负载均衡设备上启用基于真实服务器的会话保持。

Q&A案例分析（3）-客户端源IP取不到

故障现象：

服务端获取不到用户外网的IP地址，看到的都是大量来自于内网特定网段的IP地址。

原因分析：

负载均衡设备启用了用户源地址转换（SNAT）模式，修改了TCP报文中的用户源IP。

解决方案：

负载均衡设备会用用户的外网IP改写x-forwarded-for值，服务端通过获取http协议中request header头的x-forwarded-for值作为用户源IP。IIS日志通过安装插件形式显示用户源IP。

服务器负载均衡设备选型

1.价格因素

硬件设备：F5、 Citrix 、Redware 、A10

软件：LVS、Nginx、Haproxy、zen loadbalance

2.性能

4/7层吞吐量(单位bps)

4/7层新建连接数（单位CPS）

并发连接数

功能模块性能指标（ssl加速、 HTTP压缩、内存Cache）

3.满足真实和未来需求

1）如果确认负载均衡设备对所有应用的处理都是最简单的4层处理，那么理论上选择的负载均衡设备的4层性能稍高于实际性能需求即可。

2）如果确认负载均衡设备对所有应用的处理都是简单的7层处理，那么理论上选择的负载均衡设备的7层性能稍高于实际性能需求即可。

3）如果负载均衡设备处理的应用既有4层的也有7层的，建议按照7层应用的性能来考虑负载均衡设备。

4）如果确认自己的应用经过负载均衡处理时，需要复杂的4层或者7层处理，例如需要根据客户端的地址做策略性分发，需要根据tcp的内容做处理，需要根据HTTP头或者HTTP报文做处理，那么建议选择的负载均衡设备4/7层性能为真实性能需求的两倍。

5）如果负载均衡设备有混合的复杂流量处理并且还开启了一些功能模块，那么建议选择的负载均衡设备4/7层性能为真实性能需求的3倍。

6）考虑到设备需要轻载运行才能更加稳定，所以有可能的话在以上基础上再增加30%的性能。

7）如果还要满足未来几年的发展需求，在以上基础上还要留出未来发展所需要增加的性能。

8）不同负载均衡设备厂家由于不同的架构，使得某些设备在复杂环境下可能也表现的比较优秀，这个客户可以对比判断，但总体来说，以上建议适合于所有厂家的设备。■

专 题

对一边旅行一边编程的生活方式的体验和思考

□ 外刊IT评论/译

大概三个月前，我工作的公司突然倒闭了。

倒闭的原因跟这篇文章的内容毫不相干，但简言之，投资者毁约，一个开发中的产品，也就是我主要工作那个，被迫终止了。

我从没有想过会发生这种事情。一下子变成了待业，我是刚休完假回来，而且还就当前稳定的工作做了一个中长期计划。

但事情就这样发生了，而且是一夜之间，我不得不思考下一步的出路。

远程工作的想法一直吸引着我。我知道，在某种程度上我是想逃离这种朝九晚五的桎梏，带着我的笔记本，去寰球旅行，以自由编程职业者的身份做一些小项目，全面体验生活的同时还锻炼我的技术能力，不至于让自己变得生疏。

于是，突然间，我就变成了一个数字游民，没有包袱，没有贷款，没有孩子，有一点积蓄，没有近期必须要做的事情。

需要说一下，我的决定下的非常简单——这是一次绝好的做一次旅行的机会，我一直盼望着的。

我的旅游线路的选择非常的好理解，数个原因最终导致了我买了一张去往曼谷的机票。

东南亚自由港！

我到欧洲一些城市旅游过，所以我去世界里另外一部分。

我曾在美国待过一年，我很想回去看看，但我迫不及待的想开始我的旅程，而对一个波兰人说，美国签证既昂贵又难拿到。

澳大利亚对于做长期旅游来说花费实在太高，尤其是像我这种情况，我喜欢潜水、冲浪和其它一些户外运动，这些运动即使在一些“便宜”的国家里也是价格不菲，更别提在澳大利亚了。

所以，东南亚成了很明显的选择。那里是有预算限制的旅游人的圣地。网上有很多到过那里的人写的无数的博客，我想，这应该是一个人的一生中至少要经历过一次的一件事。

幸运的是，我有一些小客户，当我还在公司上班时，下班后会给他们干几个小时，虽然收入不多，但在旅途中，这也是一种相当大的经济保障。

经过了一个月的准备(打疫苗，装备，景点计划)，我和女友背起背包(42升的Northface Duffels包——高度推荐，虽然不是做长途旅行的最佳选择)，启程飞向曼谷，开始了我们泰国，越南，老挝，柬埔寨的旅游计划(顺便说一下，我

们后来把线路改成了泰国，越南，柬埔寨和印度尼西亚，这是另外一个话题)。

现在，我已经旅行两个月了，为我在波兰的客户远程工作。这些简单的工作大多数是维护修改一些现有的网站，但我后来想办法接了一个大一点的Rails项目，这个需要我从头开发。

目前为止，这是一次非常值得的、有启发的(尽管不是在技术方面)的体验，但也显露出一些小的问题，我想在这里分享给大家。

主要的，我发现我需要反复面对两个非常重要的问题：

1) 项目类型限制，在旅途中你可以做的项目；

2) 对不依赖办公地点的错误认识；

只能做一些小的简单的项目

这对于我来说是巨大的不利。起初我觉得做一些小项目能让我保持漂泊状态就行了，但现在的现实情况是，我越来越有一种没有成就和浪费时间的感觉。

没错，在泰国帕岸岛的小屋里打开你的笔记本电脑，一边跟客户网上交流一边开发项目，旁边就是海滨，这很有趣。你做完工作，跳进30度左右的海水里，更美妙的，你可以自由的潜水到数个珊瑚礁里，但这需要另付费。

我并不是真正的认为长时间的旅游是一种浪费时间。旅途中你能获得很多的见识和机会，但事情往往是不能兼得。

如果你喜欢挑战，需要不断的在你的技术领域中取得自身发展，可在旅途中你很难达到这方面的平衡。

当然，这取决于你开发的项目的类型和实现中面对的技术问题，但说老实话，当你在不断的变换地方，经常遇到不可预知的网络状况，渴望最大限度的享受旅游中的快乐，你根本无法去做那些有难度的挑战性的工作，这些工作需要数小时的分析，讨论和复杂的编程。

旅途中有太多让你分心的事，工作后(有时是工

作中)有太多的东西吸引你去观看。

我非常理解，有些人会把这当作一种自制力问题，一种如何计划，如何准备的问题，但对于我来说，绝不是这样的。

我认为自己是一个非常能自控的人。我的客户满意我做的工作，我总是能按期完成任务，而且是高质量的。这是我自己定的目标。

问题不在这里。问题是，当涉及到中等复杂项目，涉及到不那么简单的编程任务时，我发现，很多在办公室里能完成的工作，在旅途中却变成了很有挑战或完全不可能的事情。

这种情况限制了我只能挑选一些容易处理，以及有富足的时间期限和只需要少量的研究的项目。

毫不隐瞒的说，还不止这些。我认为在一个由比你有经验、比你聪明的人组成的团队里工作，这对你的开发是至关重要的，作为自由职业者，只做一些小的项目、一些简单的MVC/CRUD编程工作，这就是相当于错失了很多成长和学习的机会。

但不要误会，我在忙碌的工作，每天都在努力学一些新知识，一直都在增长我作为程序员的各种技术能力。然而，在过去的两个月里，我感觉缺少把自己暴露在现实编程问题中的机会，那是我在公司工作时每天都会遇到的，那些问题是你一个人无法解决的。那些问题跟公司的大小无关。

不依赖办公地点的错觉

现在，我完全的相信远程工作是可行的，它对(a)程序员摆脱办公地点依赖和(b)减少公司开支是十分有效的途径。

但实际情况却完全不同。旅途中工作给了你一种工作地点不固定的感觉。当然，有的公司的办公地点会分布世界各地，但为了完成一个不那么简单的任务，你非常需要一个舒适的场所，好的网络连接，安静的环境，更好的一些设施，例如健身房，瑜伽室，酒吧。

还有一点很明显，在旅途中我发现，创造性较大的工作需要有一些作息规律。一周变换一个地方，工作在海滨，吊床里，咖啡馆里，酒吧里，旅馆地板上，对于度假来说这很酷，但很显然并不有助于解决困难的编程问题。

对于这些问题涉及到各种可能性和各种层面，我现在想了一些解决方案，下面就是。

1）远程工作的内容要适合你的技术和你的能力。每三个月换一种技术方向。在每一个地方要多待一段时间。要让自己沉浸在当地的文化和生活节奏中，租一个好一点的 屋子，要有一个正式的工作桌和椅子，要能确保你坐在上面能保持3个小时以上(如果你有站着工作的习惯，那就忽略这一点)，要形成规律习惯。去你喜欢的地 方，当你想换个地方时，先研究一下，然后再去。

2) 找一个真正的工作，在办公室里的工作，做你喜欢的和有挑战性的工作(事实上，一种应该这样)，争取能长期假期，去旅行，每年大概2个月。

3) 积极做一些能够获得临时收入的wordpress网站和旅游网站相关的活，这能给你带来很多的收入，能让你旅游个够。

边注：

◆ 我不喜欢“数字游民(digital nomad)”这个词来形容自己。我认为用douchy这个词更合适。

◆ 如果你有兴趣，请到airseasummit.com上关注我的探险之旅。 ■

在Java中使用启发式搜索更快地解决问题

本文作者展示了他们如何成功用 Java 实现了最广为使用的启发式搜索算法。他们的解决方案利用一个替代的 Java 集合框架，并使用最佳实践来避免过多的垃圾收集。

了解启发式搜索领域及其在人工智能上的应用。本文作者展示了他们如何成功用 Java 实现了最广为使用的启发式搜索算法。他们的解决方案利用一个替代的 Java 集合框架，并使用最佳实践来避免过多的垃圾收集。

通过搜寻可行解决方案空间来解决问题是人工智能中一项名为状态空间搜索 的基本技术。 启发式搜索 是状态空 间搜索的一种形式，利用有关一个问题的知识来更高效地查找解决方案。启发式搜索在各个领域荣获众多殊荣。在本文中，我们将向您介绍启发式搜索领域，并展示 如何利用 Java 编程语言实现 A*，即最广为使用的启发式搜索算法。启发式搜索算法对计算资源和内存提出了较高的要求。我们还将展示如何避免昂贵的垃圾收集，以及如何利用一个替代的高 性能 Java 集合框架 (JCF)，通过这些改进 Java 实现。本文的所有代码都可以从 下载 部分获得。

启发式搜索

计算机科学中的许多问题可用一个图形数据结构表示，其中图形中的路径表示潜在的解决方案。查找最优解决方案需要找到一个最短路径。例如，以自主视频游戏角色为例。角色做出的每个动作都与图形中的一个边缘相对应，而且角色的目标是找到最短路径，与对手角色交手。

深度优先 搜索和广度优先 搜索等算法是流行的图形遍历算法。但它们被视为非启发式 算法，而且常常受到它们可以解决的问题规模的严格限制。此外，不能保证深度优先搜索能找到最优解决方案（或某些情况下的任何解决方案），可以保证广度优先搜索仅能在特殊情况下找到最优解决方案。相比之下，启发式搜索是一种提示性 搜索，利用有关一个问题的知识，以启发式 方式进行编码，从而更高效地解决问题。启发式搜索可以解决非启发式算法无法解决的很多难题。

视频游戏寻路是启发式搜索的一个受欢迎的领域，它还可以解决更复杂的问题。2007 年举行的无人驾驶汽车比赛 “DARPA 城市挑战赛” 的优胜者就利用了启发式搜索来规划平坦的、直接的可行使路线。启发式搜索在自然语言处理中也有成功应用，它被用于语音识别中的文本和堆栈解码句法解析。它 在机器人学和生物信息学领域也有应用。与传统的动态编程方法相比较，使用启发式搜索可以使用更少的内存更快地解决多序列比对 (Multiple Sequence Alignment, MSA)，这是一个经过深入研究的信息学问题。

通过 Java 实现启发式搜索

Java 编程语言不是实现启发式搜索的一种受欢迎的选择，因为它对内存和计算资源的要求很高。出于性能原因，C/C++ 通常是首选语言。我们将证明 Java 是实现启发式搜索的一种合适的编程语

言。我们首先表明，在解决受欢迎的基准问题集时，A* 的 textbook 实现确实很缓慢，并且会耗尽可用内存。我们通过重访一些关键实现细节和利用替代的 JCF 来解决这些性能问题。

很多这方面的工作都是本文作者合著的一篇学术论文中发表的作品的一个扩展。尽管原作专注于 C/C++ 编程，但在这里，我们展示了适用于 Java 的许多同样的概念。

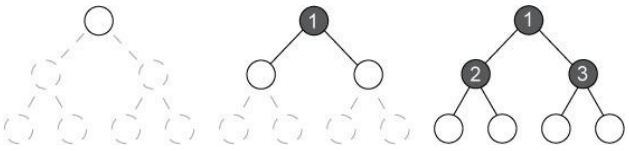
广度优先搜索

熟悉广度优先搜索（一个共享许多相同概念和术语的更简单的算法）的实现，将帮助您理解实现启发式搜索的细节。我们将使用广度优先搜索的一个以代理为中心 的视图。在一个以代理为中心的视图中，代理据说处于某种状态，并且可从该状态获取一组适用的操作。应用操作可将代理从其当前状态转换到一个新的后继 状态。该视图适用于多种类型的问题。

广度优先搜索的目标是设计一系列操作，将代理从其初始状态引导至一个目标状态。从初始状态开始，广度优先搜索首先访问最近生成的状态。所有适用的操作在每个访问状态都可以得到应用，生成新的状态，然后该状态被添加到未访问状态列表（也称为搜索的前沿）。访问状态并生成所有后继状态的过程被称为扩展 该状态。

您可以将该搜索过程看作是生成了一个树：树的根节点表示初始状态，子节点由边缘连接，该边缘表示用于生成它们的操作。图 1 显示该搜索树的一个图解。白圈表示搜索前沿的节点。灰圈表示已展开的节点。

图 1. 二叉树上的广度优先搜索顺序



搜索树中的每一个节点表示某种状态，但两个独特的节点可表示同一状态。例如，搜索树中处于不同深度的一个节点可以与树中较高层的另一个节点具有同样的状态。这些重复 节点表示在搜索问题中达到同一状态的两种不同方式。重复节点可能存在问题，因此必须记住所有受访节点。

清单 1 显示广度优先搜索的伪代码：

清单 1. 广度优先搜索的伪代码

```
1. function: BREADTH-FIRST-SEARCH(initial)
2. open ← {initial}
3. closed ← 0
4. loop do:
5.      if EMPTY(open) then return failure
6.      node ← SHALLOWEST(open)
7.      closed ← ADD(closed, node)
8.      for each action in ACTIONS(node)
9.           successor ← APPLY(action, node)
10.               if successor in closed then
  continue
11.               if GOAL(successor) then return
  SOLUTION(node)
12.               open ← INSERT(open, successor)
```

在 清单 1 中，我们将搜索前沿保留在一个 open 列表（第 2 行）中。将访问过的节点保留在 closed 列表（第 3 行）中。closed 列表有助于确保我们不会多次重访任何节点，从而不会重复搜索工作。仅当一个节点不在 closed 列表中时才能将其添加到前沿。搜索循环持续至 open 列表为空

或找到目标为止。

在 图 1 中，您可能已经注意到，在移至下一层之前，广度优先搜索会访问搜索树的每个深度层的所有节点。在所有操作具有相同成本的问题中，搜素树中的所有边缘具有相同的权重，这样可保证广度优先搜索能找到最优解决方案。也就是说，生成的第一个目标在从初始状态开始的最短路径上。

在某些域中，每个操作有不同的成本。对于这些域，搜索树中的边缘具有不统一的权重。在这种情况下，一个解决方案的成本是从根到目标的路径上所有边缘权重的总和。对于这些域，无法保证广度优先搜索能找到最优解决方案。此外，广度优先搜索必须展开树的每个深度层的所有节点，直至生成目标。存储这些深度层所需的内存可能会快速超过最现代的计算机上的可用内存。这将广度优先搜索限制于很窄的几个小问题。

Dijkstra 的算法是广度优先搜索的一个扩展，它根据从初始状态到达节点的成本对搜索前沿上的节点进行排序（排列 open 列表）。不管操作成本是否统一（假设成本是非负值），它都确保可以找到最优解决方案。然而，它必须访问成本少于最优解决方案的所有节点，因此它被限制于解决较少的问题。下一节将描述一个能解决大量问题的算法，该算法能大幅减少查找最优解决方案所需访问的节点数量。

A* 搜索算法

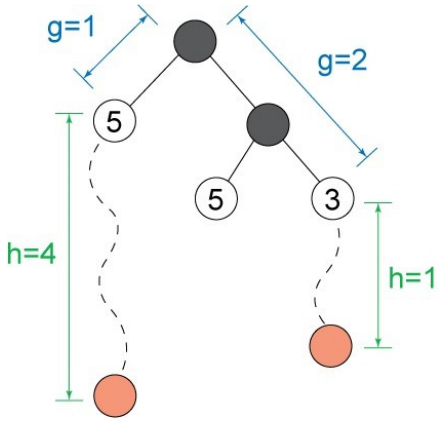
A* 算法或其变体是最广为使用的启发式搜索算法之一。可以将 A* 看作是 Dijkstra 的算法的一个扩展，它利用与一个问题有关的知识来减少查找一个解决方案所需的计算数量，同时仍然保证最优的解决方案。A* 和 Dijkstra 的算法是最佳优先 图形遍历算法的典型示例。它们是最佳优先算法，是因为他们首先访问最佳的节点，即出现在通往目标的最短路径上的那些节点，直至找到一个解决方案。对于许多问题，找到最佳解决方案至关重要，这是让 A* 这样的算法如此重要的原因。

A* 与其他图形遍历算法的不同之处在于使用了启发式估值。启发式估值是有关一个问题的一些知识（ 经验法则），该知识能让您做出更好的决策。在搜索算法的上下文中，启发式估值 有具体的含义：估算从特定节点到一个目标的成本的一个函数。A* 可以利用启发式估值来决定哪些节点是最应该访问的，从而避免不必要的计算。A* 尝试避免访问图形中几乎不通向最优解决方案的节点，通常可用比非启发式算法更少的内存快速找到解决方案。

A* 确定最应该访问哪些节点的方式是为每个节点计算一个值（我们将其称为 f 值），并根据该值对 open 列表进行排序。f 值是使用另外两个值计算出来的，即节点的 g 值 和 h 值。一个节点的 g 值是从初始状态到达一个节点所需的所有操作的总成本。从节点到目标的估算成本是其 h 值。这一估算值是启发式搜索中的启发式估值。f 值最小的节点是最应该访问的节点。

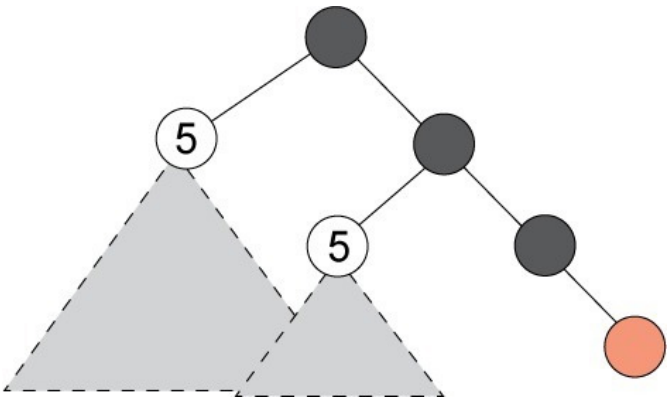
图 2 展示该搜索过程：

图 2. 基于 f 值的 A* 搜索顺序



在 图 2 的示例中，前沿有三个节点。有两个节点的 f 值是 5，一个节点的 f 值是 3。接下来展开 f 值最小的节点，该节点直接通往一个目标。这样一来 A* 就无需访问其他两个节点下的任何子树，如图 3 所示。这使得 A* 比广度优先搜索等算法要高效得多。

图 3. A* 不必访问 f 值较高的节点下的子树



如果 A* 使用的启发式估值是可接受的，那么 A* 仅访问找到最优解决方案所需的节点。为此 A* 很受欢迎。没有其他算法能用可接受的启发式估值，通过访问比 A* 更少的节点保证找到一个最优解决方案。要让启发式估算成为可接受的，它必须是一个下限值：一个小于或等于到达目标的成本的值。如果启发满足另一个属性，即一致性，那么将首次通过最优路径生成每个状态，而且该算法可以更高效地处理重复节点。

与上一节的广度优先搜索一样，A* 维护两个数据结构。已生成但尚未访问的节点存储在一个 open 列表 中，而且访问的所有标准节点都存储在一个 closed 列表 中。这些数据结构的实现以及使用它们的方式对性能有很大的影响。我们将在后面的一节中对此进行详细探讨。清单 2 显示 textbook A* 搜索的完整伪代码。

清单 2. A* 搜索的伪代码

```
1. function: A*-SEARCH(initial)
2. open ← {initial}
3. closed ← 0
4. loop do:
5.      if EMPTY(open) then return failure
6.      node ← BEST(open)
7.      if GOAL(node) then return
   SOLUTION(node)
8.      closed ← ADD(closed, node)
9.      for each action in ACTIONS(node)
10.             successor ← APPLY(action, node)
11.             if  successor in open or successor
   in closed
12.                 IMPROVE(successor)
13.             else
14.                 open ← INSERT(open,
   successor)
```

在 清单 2 中，A* 从 open 列表中的初始节点入手。在每次循环迭代中，open 列表上的最佳节点被删除。接下来，open 上最佳节点的所有适用操作被应用，生成所有可能的后继节点。对于每个后继节点，我们将通过检查确认它表示的状态是否已被访问。如果没有，则将其添加到 open 列表。如果它已经被访问，则需要通过一个更好的路径确定我们是否达到了这一状态。如果是，则需要将该节点放在 open 列表上，并删除次优的节点。

我们可以使用有关要解决的问题的两个假设简化这一段伪代码：我们假设所有操作有相同的成本，而且我们有可接受的、一致的启发式估值。因为启发式估值是一致的，且域中的所有操作具有相

同的成本，那么我们永远无法通过一个更好的路径重访一个状态。结果还表明，对于一些域，在 open 列表中放置重复节点比每次生成新节点时检查是否有重复节点更高效。因此，我们可以通过将所有新后继节点附加到 open 列表来简化实现，不管它们是否已经被访问。我们通过将 清单 2 中的最后四行组合为一行来简化伪代码。我们仍然需要避免循环，因此在展开一个节点之前，必须检查是否有重复节点。我们可以省略掉 IMPROVE 函数的细节，因为在简化版本中不再需要它。清单 3 显示简化的伪代码：

清单 3. A* 搜索的简化伪代码

```
1. function: A*-SEARCH(initial)
2. open ← {initial}
3. closed ← 0
4. loop do:
5.      if EMPTY(open) then return failure
6.      node ← BEST(open)
7.      if node in closed continue
8.      if GOAL(node) then return
   SOLUTION(node)
9.      closed ← ADD(closed, node)
10.          for each action in ACTIONS(node)
11.              successor ← APPLY(action, node)
12.              open ← INSERT(open, successor)
```

A* 的 Java textbook 实现

本节我们将介绍如何基于 清单 3 中简化的伪代码完成 A* 的 Java textbook 实现。您会看到，这一实现无法解决 30GB 内存限制下的一个标准启发式搜索基准。

我们希望我们的实现尽量大众化，因此我们首先定义了一些接口来提取 A* 要解决的问题。我们想通过 A* 解决的任何问题都必须实现 Domain接口。Domain 接口提供具有以下用途的方法：

- 查询初始状态
- 查询一个状态的适用操作
- 计算一个状态的启发式估值
- 生成后继状态

清单 4 显示了 Domain 接口的完整代码：

清单 4. Domain 接口的 Java 源代码

```
1. public interface Domain<T> {
2.   public T initial();
3.   public int h(T state);
4.   public boolean isGoal(T state);
5.   public int numActions(T state);
6.   public int nthAction(T state, int nth);
7.   public Edge<T> apply (T state, int op);
8.   public T copy(T state);
9. }
```

A* 搜索为搜索树生成边缘和节点对象，因此我们需要 Edge 和 Node 类。每个节点包含 4 个字段：节点表示的状态、对父节点的引用，以及节点的 g 和 h 值。清单 5 显示 Node 类的完整代码：

清单 5. Node 类的 Java 源代码

```
1. class Node<T> {
2.   final int f, g, pop;
3.   final Node parent;
4.   final T state;
```

```
5.  private Node (T state, Node parent, int
    cost, int pop) {
6.    this.g = (parent != null) ? parent.g+cost
    : cost;
7.    this.f = g + domain.h(state);
8.    this.pop = pop;
9.    this.parent = parent;
10.         this.state = state;
11.       }
12.     }
```

每个边缘有三个字段：边缘的成本或权重、用于为边缘生成后继节点的操作，以及用于为边缘生成父节点的操作。清单 6 显示了 Edge 类的完整代码：

清单 6. Edge 类的 Java 源代码

```
1. public class Edge<T> {
2.  public int cost;
3.  public int action;
4.  public int parentAction;
5.  public Edge(int cost, int action, int
    parentAction) {
6.    this.cost = cost;
7.    this.action = action;
8.    this.parentAction = parentAction;
9.  }
10.     }
```

A* 算法本身会实现 SearchAlgorithm 接口，而且仅需要Domain和Edge接口。SearchAlgorithm接口仅提供一个方法来执行具有指定初始状态的搜索。search() 方法返回 SearchResult 的一个实例。SearchResult 类提供搜索统计。SearchAlgorithm 接口的定义如清单 7 所示：

清单 7. SearchAlgorithm 接口的 Java 源代码

```
1. public interface SearchAlgorithm<T> {
2.  public SearchResult<T> search(T state);
3. }
```

用于 open 和 closed 列表的数据结构的选择是一个重要的实现细节。我们将使用 Java 的 PriorityQueue 实现 open 列表。PriorityQueue 是一个平衡的二进制堆的实现，包含用于元素入列和出列的 O(log n) 时间、用于测试一个元素是否在队列中的线性时间，以及用于访问队列头的约束时间。二进制堆是实现 open 列表的一个流行数据结构。稍后您会看到，对于一些域，可以使用一个名为桶优先级队列 的更高效的数据结构来实现 open 列表。

我们必须实现 Comparator 接口让 PriorityQueue 合理地对节点进行排序。对于 A* 算法，我们需要根据 f 值对每个节点排序。在许多节点的 f 值相同的域中，一个简单的优化是通过选择 g 值较高的节点来打破平局。试着花点时间说服自己为何以这种方式打破平局能提高 A* 的性能（提示：h 是一个估算值；而 g 不是）。清单 8 包含完整的 Comparator 实现代码：

清单 8. NodeComparator 类的 Java 源代码

```
1. class NodeComparator implements
   Comparator<Node> {
2.  public int compare(Node a, Node b) {
```

```
3.    if (a.f == b.f) {
4.      return b.g - a.g;
5.    }
6.    else {
7.      return a.f - b.f;
8.    }
9.  }
10.     }
```

我们需要实现的其他数据结构是 closed 列表。对此的一个明显的选择是 Java 的 HashMap 类。HashMap 类是散列表的一个实现，只要我们提供一个好的散列函数，预期会有用于检索和添加元素的恒定时间。我们必须重写负责实现域状态的类的 hashcode() 和 equals() 方法。我们将在下一节中探讨该实现。

最后我们需要实现 SearchAlgorithm 接口。为此，我们使用 清单 3 中的伪代码实现 search() 方法。清单 9 显示了 A* search() 方法的完整代码：

清单 9. A* search() 方法的 Java 源代码

```
1. public SearchResult<T> search(T init) {
2.   Node initNode = new Node(init, null, 0, 0 -1);
3.   open.add(initNode);
4.   while (!open.isEmpty() && path.
   isEmpty()) {
5.     Node n = open.poll();
6.     if (closed.containsKey(n.state))
   continue;
7.     if (domain.isGoal(n.state)) {
8.       for (Node p = n; p != null; p = p.
   parent)
9.         path.add(p.state);
10.          break;
11.        }
12.        closed.put(n.state, n);
13.        for (int i = 0; i < domain.numActions(n.
   state); i++) {
14.          int op = domain.nthAction(n.state, i);
15.          if (op == n.pop) continue;
16.          T successor = domain.copy(n.state);
17.          Edge<T> edge = domain.
   apply(successor, op);
18.          Node node = new Node(successor, n,
   edge.cost, edge.pop);
19.          open.add(node);
20.        }
21.      }
22.      return new SearchResult<T>(path,
   expanded, generated);
23.    }
```

要评估我们的 A* 实现，需要对一个问题运行该实现。在下一节中，我们将描述一个用于评估启发式搜索算法的流行域。该域中的所有操作都具有相同的成本，而且我们使用的启发式估值是可接受的，因此我们的简化实现是足够的。

15 puzzle 基准

本文中，我们侧重于一个名为 15 puzzle 的 toy 域。这个简单的域有易于理解的属性，是评估启发式搜索算法的一个标准基准。（有人将这些拼图称为 AI 研究的 “果蝇”。）15 puzzle 是一种滑块拼图，在 4×4 网格上排列 15 个滑块。一个滑块有 16 个位置可供选择，总有一个位置是空

的。与空白位置相邻的滑块可从一个位置滑动 到另一个位置。其目的是滑动滑块，直至达到拼图的目标布局。图 4 显示了随机布局下的滑块拼图：

图 4. 15 puzzle 的随机布局

3	2	1	4
5		11	8
9	7	10	12
13	14	6	15

图 5 显示了目标布局下的滑块：

图 5. 15 puzzle 的目标布局

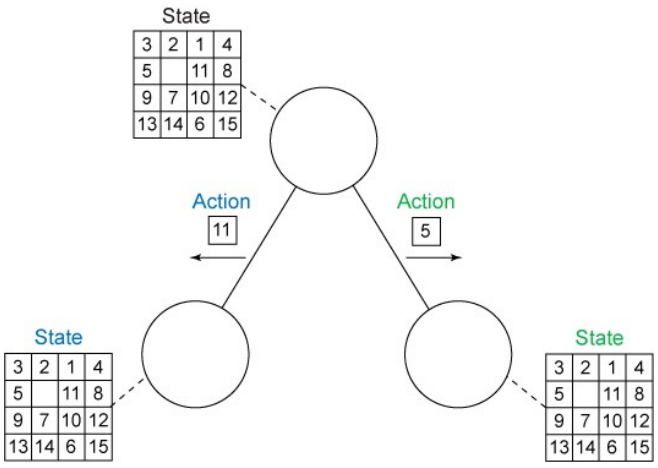
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

作为一个启发式搜索基准，我们希望通过尽量少的移动从某个初始布局开始找到该拼图的目标布局。

我们将为该域使用的启发式算法叫作曼哈坦距离 算法。一个滑块的曼哈坦距离是滑块到达目标位置所需做出的垂直和水平移动数量。要计算一个状态的启发式估值，我们需要算出拼图中所有滑块的曼哈坦距离的总和，忽略空白位置。对于任何状态，所有这些距离之和必须是达到拼图目标状态所需成本的下限值，因为您永远无法通过减少移动量将滑块移动到每个目标布局。

一开始似乎不太直观，但我们可以用图形建模 15 puzzle，将滑块的每个可能布局表示为节点。如果有一个操作将一个布局转化为另一个布局，一个边缘连接两个节点。在该域中，一个操作将滑块滑到空白区域。图 4 展示了这一搜索图：

图 6. 15 puzzle 的状态空间搜索图



有 16! 种在网格上排列 15 个滑块的可能方式，不过实际上“只有” 16!/2 = 10,461,394,944,000 种 15 puzzle 的可达布局或状态。这是因为拼图的物理约束让我们刚好达到所有可能布局的一半。为了了解该状态空间的大

小，假设我们可以用一个字节表示一种状态（这是不可能的）。为了存储整个状态空间，我们需要超过 10TB 的内存。这将远远超过最现代的计算机的内存限制。我们将展示启发式搜索如何在仅访问一小部分状态空间的同时以最佳方式解决这个难题。

运行实验

我们的实验使用 15 puzzle 的一组著名起始布局，叫 Korf 100 集合。这一集合得名于 Richard E. Korf，他发布了首批结果，表明可以使用 A* 的一个迭代加深 变体，即 IDA*，解决随机的 15 puzzle 布局。由于这些结果被发布，Korf 在其实验中使用的 100 个随机实例在无数随后的启发式搜索实验中得到重用。我们还优化了我们的实现，因而无需再使用迭代加深技术。

我们一次解决一个起始布局。每个起始布局都存储在一个独立的普通文本文件中。文件的位置在启动实验的命令行参数中指定。我们需要一个 Java 程序入口点来处理命令行参数，生成问题实例，并运行 A* 搜索。我们将这个入口点称为 TileSolver 类。

在每次运行结束时会将有关搜索性能的统计数据打印为标准输出。我们最感兴趣的统计数据是挂钟时间（wall clock time）。我们将所有运行的挂钟时间加起来，得出该基准测试的总运行时间。

本文 源代码 包含自动完成实验的 Ant 任务。您可以使用以下代码运行整个实验：

```
1. ant TileSolver.korf100
   -Dalgorithm=”efficient”
```

可以为 algorithm 指定 efficient 或 textbook。

我们还提供一个 Ant 目标来运行单一实例。例如：

```
1. ant TileSolver -Dinstance=”12”
   -Dalgorithm=”textbook”
```

而且我们提供一个 Ant 目标来运行基准测试子集，其中不包括三个最难的实例：

```
1. ant TileSolver.korf97 -Dalgorithm=”textbook”
```

很可能您的计算机没有足够的内存来使用 textbook 实现完成整个实验。为了避免存储交换，您应当谨慎地限制 Java 进程可用的内存量。如果您要在 Linux 上运行该实验，可以使用像 ulimit 这样的 shell 命令来设置活动 shell 的内存限制。

一开始没有成功

表 1 显示我们使用的所有技术的结果。Textbook A* 实现的结果在第一行。（在后面的章节中我们描述了 Packed states 和 HPPC 及其相关结果。）

表 1. 解决 97 个 Korf 100 15 puzzle 基准测试实例的三个 A* 变体的结果

算法	最大内存使用量	总运行时间
Textbook	25GB	1,846 秒
Packed states	11GB	1,628 秒
HPPC	7GB	1,084 秒

textbook 实现无法解决所有测试实例。我们未能解决三个最难的实例，而且对于我们可以解决的实例，我们花了超过 1,800 秒的时间。这些结果不算很好，因为 C/C++ 中最好的实现可在不到 600 秒的时间内解决 100 个实例。

由于内存约束我们未能解决最难的那三个实例。在每一次搜索迭代中，从 open 列表中删除一个节点并展开它通常会导致生成更多的节点。随着所生成节点数量的增加，在 open 列表中存储它们所需的内存量也在增加。但是，这一内存需求不是 Java 实现所特有的；同等的 C/C++ 实现也会失败。

Burns 等人在其论文中表明，A* 搜索的一个高效 C/C++ 实现可以用不到 30GB 的内存解决该基准测试，因此我们还没打算放弃 Java A* 实现。我们还可以应用后续章节中讨论的其他技术，更高效地使用内存。最终得出一个能够快速解决整个基准测试的高效的 A* Java 实现。

包装状态

在使用像 VisualVM 这样的探查器检查 A* 搜索的内存使用情况时，我们看到所有内存都被 Node 类占用，更直接地说由 TileState 类占用。为了降低内存使用量，我们需要重访这些类的实现。

每个滑块状态必须存储所有 15 个滑块的位置。为此我们将每个滑块的位置存储在一个包含 15 个整数的数组中。我们可以更简明地表示这些位置，将它们包装成一个 64 位的整数（在 Java 中是 long）。当我们需要在 open 列表中存储一个节点时，可以仅存储状态的包装表示。这么做会为每个节点节省 52 个字节。要解决基准测试中最难的实例，需要存储大约 5.33 亿个节点。通过包装状态表示，我们可以节省 25GB 的内存！

为了维护我们的实现的一般性，我们需要扩展 SearchDomain 接口，使用包装和拆装状态的方法。在 open 列表上存储一个节点之前，我们将生成状态的一个包装表示，并将该包装表示存储在 Node 类（而非状态指针）中。当我们需要生成一个节点的后续节点时，只需拆装状态。清单 10 显示了 pack() 方法的实现：

清单 10. pack() 方法的 Java 源代码

```
1. public long pack(TileState s) {
2.   long word = 0;
3.   s.tiles[s.blank] = 0;
4.   for (int i = 0; i < Ntiles; i++)
5.     word = (word << 4) | s.tiles[i];
6.   return word;
7. }
```

清单 11 显示了 unpack() 方法的实现：

清单 11. unpack() 方法的 Java 源代码

```
1. public void unpack(long packed, TileState
   state) {
2.   state.h = 0;
3.   state.blank = -1;
4.   for (int i = numTiles - 1; i >= 0; i--) {
5.     int t = (int) packed & 0xF;
6.     packed >>= 4;
7.     state.tiles[i] = t;
8.     if (t == 0)
9.       state.blank = i;
10.          else
11.            state.h += md[t][i];
12.       }
13.     }
```

由于包装表示是状态的一种规范形式，我们可以将包装表示存储在 closed 列表中。我们无法将基元存储在 HashMap 类中。需要将它们包装在

Long 类的一个实例中。

表 1 中的第二行显示使用包装状态表示运行实验的结果。通过使用包装状态表示，我们将内存使用量减少了 55%，并缩短了运行时间，但我们仍然无法解决整个基准测试。

Java 集合框架的问题

如果您认为将每个包装状态表示包装在一个 Long 实例中似乎需要很大开销，那么您说得没错。它浪费内存，而且可能导致过度的垃圾收集。JDK 1.5 增加了对 autoboxing 的支持，autoboxing 会自动转换其对象表示的基元值（long 到 Long），反之亦然。对于大型集合，这些转换可能会降低内存和 CPU 性能。

JDK 1.5 还引入了 Java 泛型：一个通常相对于 C++ 模板的特性。Burns 等人表明，C++ 模板在实施启发式搜索时提供了巨大的性能优势。泛型不提供这种优势。泛型是使用 type-erasure 实现的，这会在编译时删除（消除）所有类型信息。因此，必须在运行时检查类型信息，对于大型集合来说这会导致性能问题。

HashMap 类的实现揭露出一些其他内存开销。HashMap 存储包含内部 HashMap$Entry 类实例的一个数组。每当我们将一个元素添加到 HashMap 时，都会有一个新条目被创建并添加到数组。该条目类的实现通常包含三个对象引用和一个 32 位整数的引用，每个条目总共 32 个字节。由于 closed 列表中有 5.33 亿个节点，我们将有超过 15GB 的内存开销。

接下来，我们将介绍 HashMap 类的一个替代方法，该方法支持我们通过直接存储基元进一步减少内存使用。

高性能的原生集合

由于我们现在仅在 closed 列表中存储基元，我们可以利用高性能的原生集合（High Performance Primitive Collections，HPPC）。HPPC 是一个替代的集合框架，支持您直接存储基元值，而没有 JCF 带来的所有那些开销。与 Java 泛型相比，HPPC 使用了一种类似 C++ 模板的技术，在编译时生成每个集合类和 Java 基元类型的独立实现。这样一来，在将基元值存储到一个集合中时就无需使用 Long 和Integer 这样的类包装基元值。其另外一个作用是可以避免对于 JCF 来说必需的大量强制转换。

另外还有用于存储基元值的其他 JCF 替代方法。Apache Commons Primitive Collections 和 fastutils 就是两个不错的示例。不过，我们认为 HPPC 的设计在实现高性能算法时有一个显著的优势：它为每个集合类公开内部数据存储。直接访问这一存储可以实现许多优化。例如，如果我们想将 open 或 closed 列表存储在磁盘上，直接访问底层数据数组比通过一个迭代器间接访问数据更有效。

我们可以修改 A* 实现，对 closed 列表使用 LongOpenHashSet 类的一个实例。我们需要做的更改相当简单。我们不再需要重写 state class 的hashcode 和 equals 方法，因为我们仅需存储基元值。closed 列表是一个集合（它不包含重复元素），因此我们仅需要存储值，而无需存储键/值对。

表 1 中的第三行显示了用 HPPC 取代 JCF 后运行实验的结果。凭借 HPPC，我们将内存使用量减少了 27%，将运行时间减少了 33%。

既然内存总共减少了 82%，我们就可以在内存

约束内解决整个基准测试了。结果显示于表 2 中的第一行：

表 2. 解决全部 100 个 Korf 100 基准测试实例的三个 A* 变体的结果

算法	最大内存使用量	总运行时间
HPPC	30GB	1,892 秒
嵌套桶队列	30GB	1,090 sec
避免垃圾收集	30GB	925 秒

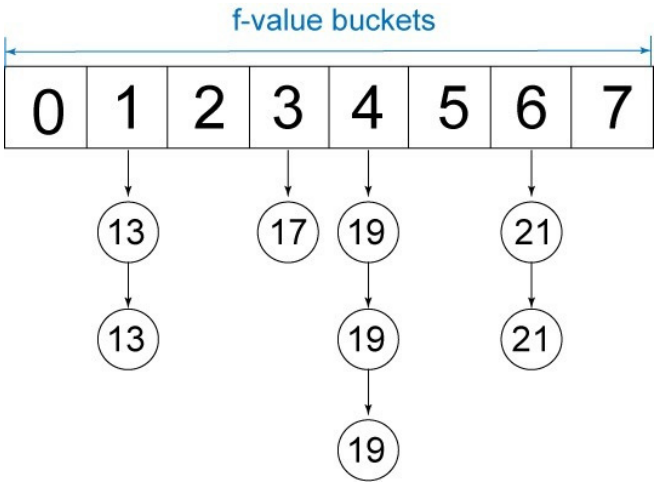
通过 HPPC，我们可以用 30GB 内存解决全部 100 个实例，但所用时间超过 1,800 秒。表 2 中的其他结果反映了我们通过改进其他重要数据结构加速实现的方式：open 列表。

PriorityQueue 的问题

每次我们将一个元素添加到 open 列表时，都需要再次排队。PriorityQueue 有 O(log(n)) 入列和出列操作时间。涉及到排序时，PriorityQueue 是高效的，但显然是不自由的，特别在 n 值较大时。记得对于最难的问题实例，我们添加了超过 5 亿个节点到 open 列表。此外，由于我们的基准测试问题中的所有操作都具有相同的成本，可能的 f 值的范围较小。因此使用 PriorityQueue 的优势与开销相比可能得不偿失。

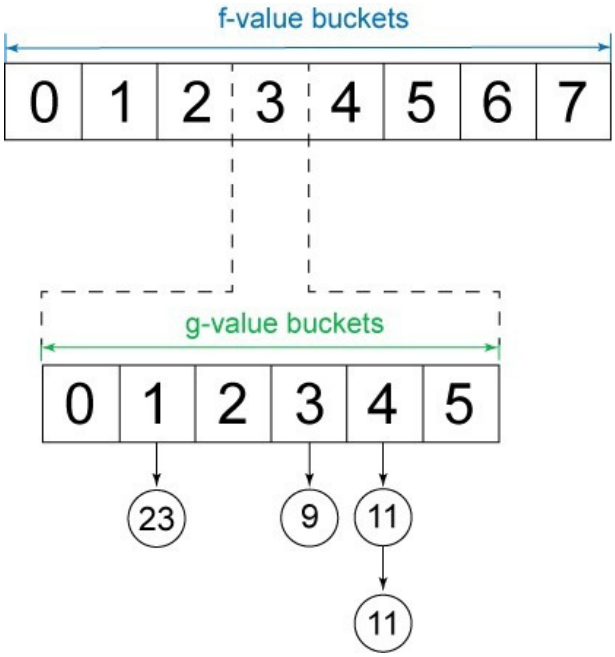
另一个替代方法是使用基于桶的优先级队列。假设我们域中的操作成本在一个狭窄的值域内，我们可以定义一个固定的存储桶范围：每个 f 值一个存储桶。当我们生成一个节点时，只需将它放在与 f 值对应的存储桶中。当我们需要访问队列头时，可以首先从 f 值最小的存储桶看起，直至我们找到一个节点。这种数据结构叫作 1 级桶优先级队列，它支持恒定入列和出列操作。图 7 展示了这一数据结构：

图 7. 1 级桶优先级队列



精明的读者会注意到，如果我们实现这里描述的 1 级桶优先级队列，就失去了使用 g 值打破节点间关系的能力。您之前应当已经意识到，以这种方式打破关系是一种值得的优化。为了维持这一优化，我们可以实现一个嵌套 桶优先级队列。1 级存储桶用于表示 f 值的范围，嵌套级别用于表示 g 值的范围。图 8 展示了这一数据结构：

图 8. 嵌套桶优先级队列



现在我们可以更新我们的 A* 实现，为 open 列表使用一个嵌套桶优先级队列。嵌套桶优先级队列的完整实现可在本文源代码中包含的 BucketHeap.java 文件中找到（参见 下载 部分）。

表 2 的第二行显示使用嵌套桶优先级队列运行实验的结果。通过使用一个嵌套桶优先级队列，而非 PriorityQueue，我们将运行时间缩短了将近 58%，所用时间是 1,000 多秒。我们可以再做一件事来缩短运行时间。

避免垃圾收集

垃圾收集常被视为 Java 中的一个瓶颈。有许多关于 JVM 中垃圾收集调优的优秀文章可应用于这里，因此我们不会详细讨论该主题。

A* 通常生成许多短暂的状态和边缘对象并招致大量昂贵的垃圾收集。通过重用对象，我们可以减少所需的垃圾收集量。为此我们可以做一些简单的更改。在 A* 搜索循环的每一次迭代中，我们分配一个新边缘和一个新状态（对于最难的问题是 5.33 亿个节点）。与其每次分配新对象，我们可以在所有循环迭代中重用相同的状态和边缘对象。

为了拥有可重用的边缘和状态对象，我们需要修改 Domain 接口。 与其让 apply() 方法返回 Edge 的一个实例，我们需要提供自己的实例，该实例通过调用 apply() 加以修改。对 edge 的更改不是递增的，因此在将其传递给 apply() 之前，我们无需担心哪些值存储在 edge 中。不过，apply() 对 state 对象所做的更改是 递增的。为了合理生成所有可能的后继状态，而无需复制状态，我们需要一种撤销所做更改的方式。为此，我们必须扩展 Domain 接口，得到一个 undo() 方法。清单 12 显示 Domain 接口更改：

清单 12. 更新的 Domain 接口

```
1. public interface Domain<T> {
2.   ...
3.   public void apply(T state, Edge<T> edge,
   int op);
4.   public void undo(T state, Edge<T> edge);
5.   ...
6. }
```

表 2 中的第三行显示最终实验的结果。通过循环利用我们的状态和边缘对象，我们可以避免昂贵的垃圾收集，并将运行时间缩短超过 15%。利用我们高效的 A* Java 实现，我们现在可以仅用 30GB 的内存在 925 秒内解决整个基准测试。鉴于最好的 C/C++ 实现需要 27GB 内存且花费 540 秒，这个结果已经很好了。我们的 Java 实现仅比 C/C++ 实现慢 1.7 倍，且需要大约同样的内存量。

结束语

在本文中，我们向您介绍了启发式搜索。我们提出了 A* 算法，并阐述了 Java textbook 实现。我们指出该实现存在性能问题，无法在合理的时间或内存约束内解决一个标准的基准测试问题。我们利用 HPPC 和一些可降低内存使用和避免昂贵的垃圾收集的技术解决了这些问题。我们改进的实现能够在适当的时间和内存约束内解决基准测试，这证明了 Java 是实现启发式搜索算法的一个不错选择。此外，我们在本文提供的技术也可应用于许多实际 Java 应用程序。例如，在某些情况下，使用 HPPC 可以立即提高存储大量基元值的任何 Java 应用程序的性能。

下载 j-ai-code.zip **58KB**

■

封面设计：@51CTO小林

《开发月刊》电子杂志

51CTO开发频道

数据